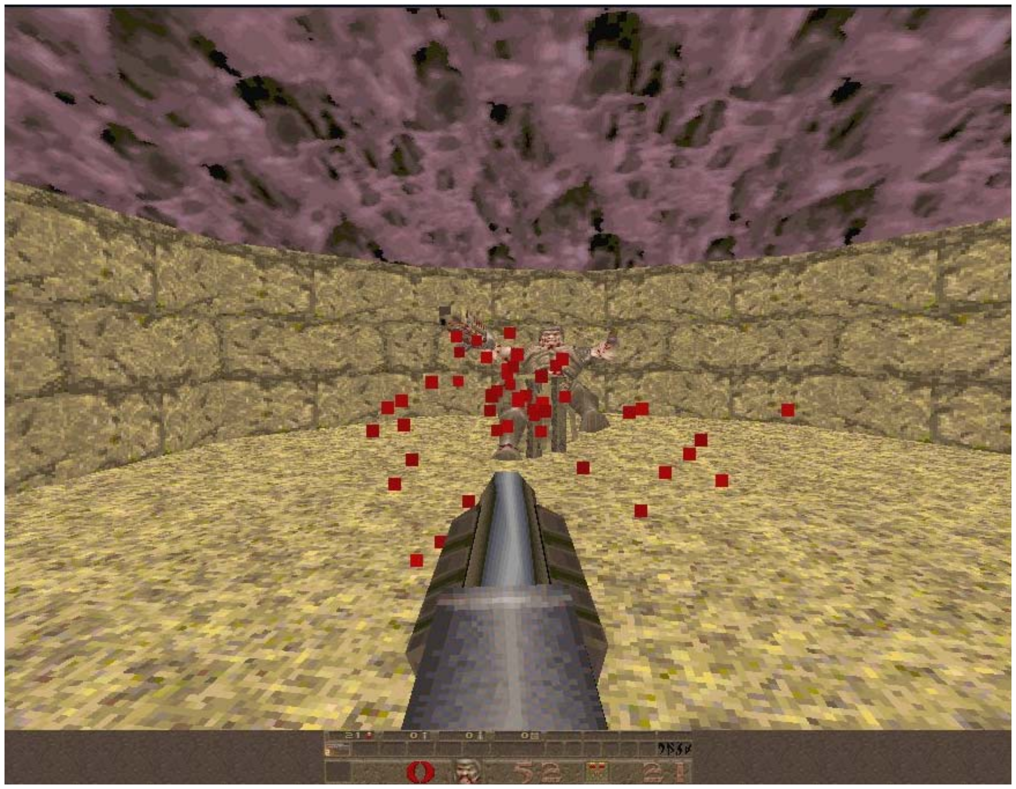
Mapwad

A Modeling Language

Final Report

Ben Smith – bhs16@columbia.edu
AvrumTilman – amt77@columbia.edu
Josh Weinberg – jmw211@columbia.edu
Ron Weiss – ronw@cs.columbia.edu



Traditionally, 3D modeling tools have taken the form of graphical editors. Advances in graphics technology have made it possible to create even more detailed models, greatly increasing the complexity of the modeling process. We have developed a computer language called Mapwad that can be used to algorithmically generate virtual environments.

Table of Contents

1. Whitepaper

1.1. Introduction

Beginning in the early 1990's with games such as Wolfenstein 3D, 3D video games, especially first person shooters (FPS), have become enormously popular. DOOM, a FPS written by *id Software*, was probably the most popular game of the first half of the 90's. The immense popularity of DOOM led to a stream of copycats trying to cash in on the FPS craze. Slightly more advanced games, such as Heretic, Duke Nukem 3D, and Descent, were released shortly after DOOM. However, *id* would not be outdone. In 1996 *id* released Quake, a far more advanced and visually appealing game. Aside from its success on the graphics front, Quake revolutionized the concept of multiplayer gaming by allowing people to play "deathmatch" games against each other over the Internet. The ability for people to challenge friends remotely also served to boost Quake's popularity. The game was so successful that even today, 7 years after its initial release, one can still find Quake game servers hosting large multiplayer games. *id* went on to release two very successful sequels to Quake, and even licensed their 3D engine technology to other game developers.

Communities revolving around modifications of *id Software*'s games have existed for years on the Internet. File formats and level specifications were widely available shortly after the release of DOOM and Quake. Enthusiasts around the world used this information to create level editors and work on creating new content for use in the games. Their efforts received a boost in 1997 when *id* released the source code for DOOM under the GNU General Public License. Two years later, they followed by releasing the Quake source as well. The tremendous online following originally created by the multiplayer capabilities of Quake still exist today precisely because of *id*'s willingness to allow fans to modify and extend their games.

Despite the interest in modifying 3D games, newcomers are put off by the large learning curve they need to cross to be able to create interesting modifications. The available tools, mostly in the form of graphical editors, are very complex to learn. The motivation for MAPWAD is to create a straightforward, yet powerful language for creating levels for games such as Quake.

1.2. Goals

The goal of this project is to design a language that will generate levels for a 3D game such as DOOM or Quake. We will be using the Quake rendering engine to represent the MAPWAP created models because of all the open source code and tools that are available. Beyond gaming, the ability to algorithmically generate 3D interactive environments is useful for any virtual reality application. Using MAPWAD, designers will be able to take advantage of algorithmic programming constructs to create such environments. In other words, MAPWAD will make a simple description yield a complex result - what used to take an hour to do by hand will be done in just a few lines of MAPWAD.

1.2.1. Ease-of-use

MAPWAD will provide a high-level approach to virtual environment design. Instead of defining polygons based on their coordinates in 3D-space, MAPWAD will describe complicated scenes as collections of objects.

1.2.2. Automation:

MAPWAD will harness the power of loops and conditionals to provide the user with the ability to generate regular structures. For example, instead of having to define every stair as a separate platform, MAPWAD will allow for the use of loops to create a staircase iteratively.

1.2.3. Flexibility:

MAPWAD will not be limited to production of video game levels. The generic nature of the modeling language will allow MAPWAD to be used for other tasks, such as: interior modeling/design, landscape modeling, or any other task that would require a 3D modeling solution.

1.2.4. Extendability:

MAPWAD will be as generic as possible. With no direct connection to a specific game, it should be possible to port MAPWAD to a new rendering engine's specifications at a later date.

1.3. Example Applications

1.3.1. Simple levels

The obvious application of MAPWAD would be as a tool for generating game levels. The most common type of game that could take advantage of MAPWAD would be a 3D first person shooter such as DOOM or Quake. MAPWAD could be used to generate the layout and dimensions of the various rooms and corridors found in a typical first person shooter game level. The appropriate MAPWAD compiler would then translate the code into a level file for a specific game engine.

1.3.2. Advanced levels

More interesting applications of MAPWAD would take advantage of its control flow abilities. A random maze algorithm could be written in MAPWAD and used to generate mazes in the form of DOOM levels. Maps could, in fact, be generated based on any mathematical function that can be visualized in two or three dimensions.

MAPWAD could also be useful for describing geometric features that would be difficult or tedious to design manually. These might include complicated architectural features such as spiral staircases.

1.3.3. Interior modeling

MAPWAD's capabilities could extend beyond DOOM level creation. Although it is not intended to replace CAD systems, MAPWAD would be an excellent tool for initial concept designs by a builder and/or floor layout specialist. MAPWAD interactive designs could be produced more quickly than hand designed versions; because the models can be viewed on a widely available and inexpensive 3D engine, clients would be able to view and explore the models on their own hardware.

1.3.4. Simulation

The carefully hand designed game world may be a source of pride to the devoted DOOM or Quake fan. Other applications of interactive virtual environments however, will have different requirements and constraints. For military training simulations, for example, it might be desirable to be able to quickly generate many different interactive environments matching certain specifications, but that are not identical. A MAPWAD program could be written which would generate new levels with a random map layout, but always matching the requirements of the mission being trained for.

1.4. Sample Code

```
//a basic four walled room
room FourWaller {
      FourWaller(length, width, height) {
            //define a four wall room...
      }

      plane wall1,wall2,wall3,floor,ceiling;
      entry door;
}

//a corridor with a door in the middle
room Corridor {
      //numDoors specifies number of doors opening out of the
middle of the corridor
      //doors[0] and doors[numDoors-1] are doors at either end of
corridor
      Corridor(length, width, height, numDoors) {
            //define a corridor...
      }

      plane ceiling,floor,wall1,wall2;
      entry doors[numDoors+2];
}

//A simple circular map
//A circular corridor forms a loop, with rooms opening off of the
//corridor segments
map_start {
      FourWaller aRoom=FourWaller(200,200,50);
```

```
        Corridor aCorridor = Corridor(400,50,60,1);

        entry first=aCorridor.doors[0];

        attach(aCorridor.doors[1],aRoom.door);

        entry prev=aCorridor.entryEnd;

        for(i=0;i < 10;i++) {
              aCorridor = Corridor(400,50,60,1);
              attach(aCorridor.entryStart,prev);

              aRoom=FourWaller(100,100,50);

              attach(aCorridor.doors[1],aRoom.door);

              prev=aCorridor.doors[2];
        }

        attach(prev,first);
}
```

2. Mapwad Tutorial

Provided you have basic programming experience this tutorial will guide you through the creation of a basic Mapwad map. A Mapwad program has two parts, global variable declarations and function definitions. The only requirements are that a map have a Map() function and that a valid MapStart value is defined somewhere in the program.

2.1. Compilation

Mapwad files are compiled using mwc, the mapwad compiler. mwc has one required parameter, the name of the source file. You can optionally include a destination file, otherwise the default file will be used. For a full listing of mwc options run mwc –h.

Once a bsp file is created, move it to the <quakedir>/id1/maps directory, run quake, and hit ` to bring up the console. Once in the console, type map <mapname> to load the map.

2.2. Basic Example

The following is the most basic program allowed in Mapwad.

```
Map()
{
      Room r = Room();
      Add(r,Wall());
      Add(r,Wall());
      Add(r,Wall());
      Add(r,Wall());
      MapStart=Location(r.Walls[0]);
}
```

This program will create a basic 4-walled room. Notice the only requirement in the program is that MapStart is given a value. Also, MapStart is never explicitly defined as a location; rather it is an implicit variable that must be given a location value for a valid map to exist.

Map() is the main function in Mapwad. When a Mapwad program is run, Map() is called first. Any other functions defined in the file must be called by Map() directly or indirectly (ie called by a function that is called by Map()).

This program also makes use of three of the four main object types in Mapwad: Room, Wall, and Location. It should be noted that this basic program relies on a number of default values set within the objects' constructors.

2.3. Basic Programming Constructs

Beyond the advanced object types Mapwad supports int, float, string, and boolean types, as well as basic control flow statements like if, else, for, while, continue, break, and return. Mapwad also supports all standard arithmetic operators. These types, statements, and operators all work similarly to how they work in Java.

So, the basic example given above could be shortened to:

```
Map()
{
      int i;
      Room r = Room();
      for(i=0;i<3;i++)
      {
            Add(r,Wall());
      }
      MapStart=Location(r.Walls[0]);
}
```

Notice, that Mapwad requires variables to be defined outside of the for statement signature. Also, unlike C/Java, all for and if statements must have { } even if they have only one statement.

2.4. Advanced Example

The following is a more advanced program that demonstrates the use of user-defined functions. Although it is not required, the defined function has default values for its parameters.

```
/*
 * There are default values given for all the function parameters
 * so any one of them can be excluded.
 */
Room regularRoom(int walls=4, float length=10,
                 string texture="GROUND1_2")
{
      Room newRoom = Room();
      int i;
      float angle = 180*(walls-2)/walls;
      Wall wall;

      for(i=0;i < walls;i++)
      {
            // the default wall name will be used
            wall=Wall(length,angle, ,texture);
            Add(newRoom,wall);
      }
      Close(newRoom);
      return newRoom;
}
Map()
{
      /*
       * r1 will be a square with sides of 10 and wall textures of "GROUND1_2"
       * r2 will be a pentagon with sides of 15 and wall textures of "GROUND1_2"
       * r3 will be a square with sides of 15 and wall textures of "GROUND1_2"
       * r4 will be a square with sides of 10 and wall textures of "GROUND1_3"
       */
      Room r1=regularRoom();
```

```
        Room r2=regularRoom(5,15);
        Room r3= regularRoom( ,15);
        Room r4=regualrRoom( , ,"GROUND1_3");

        /*
         * MapStart is 5 units away and centered with respect to
         * the first Wall of r1
         */
        MapStart=Location(r1.Walls[0], ,5);
}
```

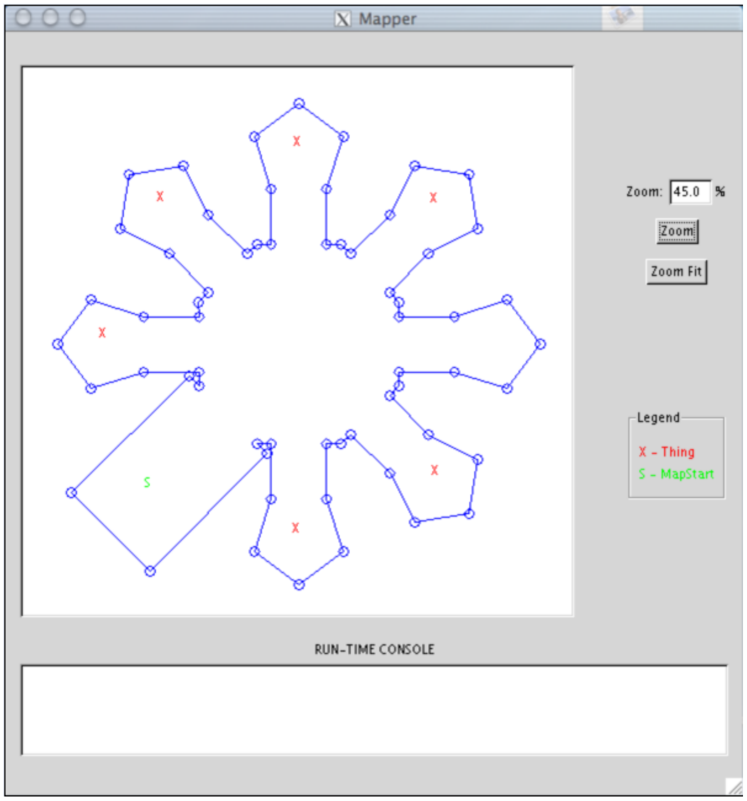
Notice also that regularRoom() was able to return a Room object just as you would have expected.

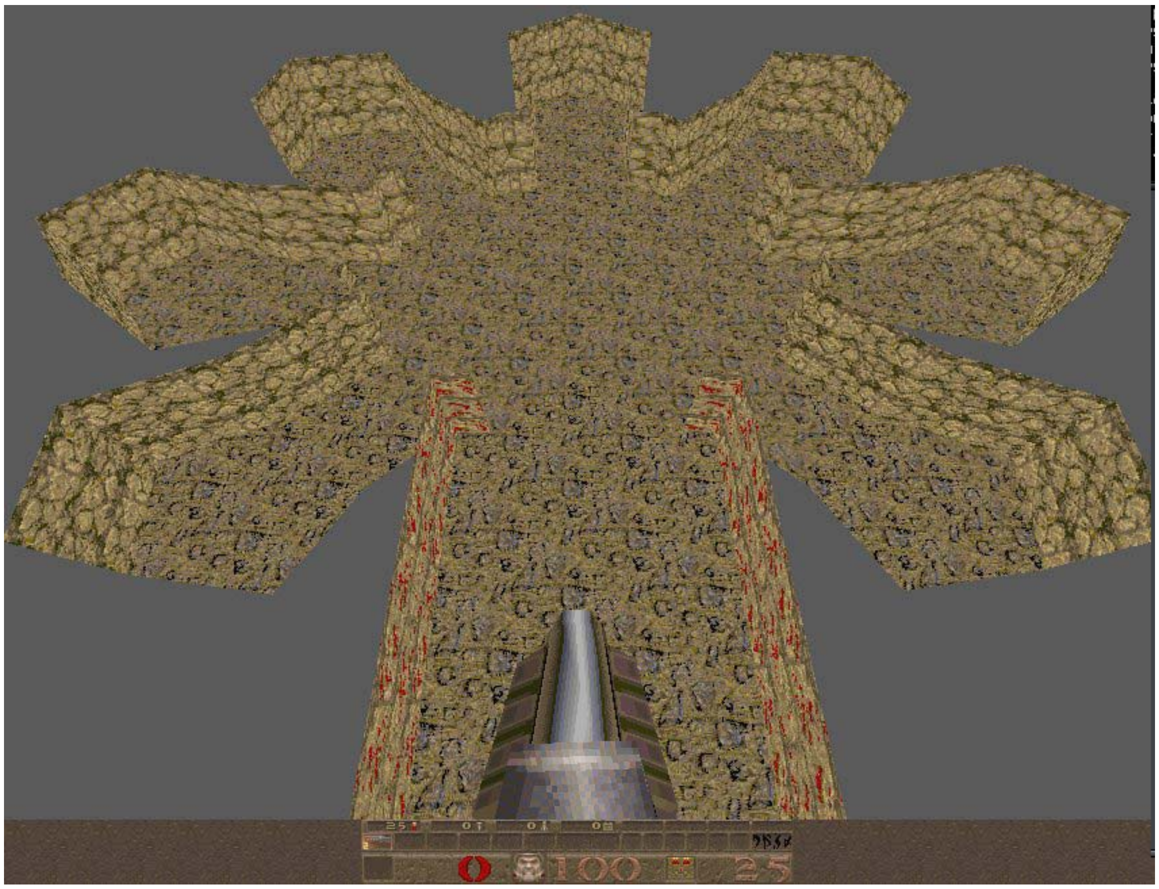
2.5. Complex Example

The following program shows more of the complex features of Mapwad.

```
/*
 * creates a regular polygon
 */
Room regularRoom(int walls=4, float length=10,
                 string texture="BRICKA2_2")
{
      Room newRoom = Room();
      int i;
      float angle = 180*(walls-2)/walls;
      Wall wall;

      for(i=0;i<walls;i++)
      {
            //the default wall name will be used
            wall=Wall(length,angle, ,texture);
            Add(newRoom,wall); // Add the wall to newRoom
      }
      Close(newRoom);
      return newRoom;
}

/*
 * main Map function
 */
Map()
{
      int i;

      // a list of possible badguys
      string[] baddies = {"monster_army", "monster_ogre",
                          "monster_zombie", "monster_dog",
                          "monster_knight", "monster_wizard",
                          "monster_ogre", "monster_demon1",
                          "monster_shambler"};

      Room oct=regularRoom(8,15); // create an octagon
      for(i=1;i<8;i++)
      {
            Room square=regularRoom(); // create a square
```

```
        Room pent=regularRoom(5); // create a pentagon

        /*
         * attach the square to a side of the octagon
         * attach the pentagon to the other side of
         * the square.
         */
        Attach(oct.Walls[i],square.Walls[0]);
        Attach(square.Walls[2],pent.Walls[0]);

        int rand=RandInt(10);

        // 70% chance of a baddie in this room
        if(rand<7)
        {
            // choose the wall for the baddie
            int w=1+RandInt(4);
            Thing(baddies[RandInt(baddies.Size)],
                Location(pent.Walls[w], ,7));
        }
    }

    //Add the start room
    Room startRoom=Room();
    Add(startRoom,Wall(,,"attach")); // name this wall attach
    Add(startRoom,Wall(30));
    Add(startRoom,Wall(,,"start")); // name this wall start
    Close(startRoom); // closes the room

    /*
     * Use the associative array property to retrieve
     * walls from the Walls array.
     */
    // attach the "attach" wall to the octagon
    Attach(startRoom.Walls["attach"],oct.Walls[0]);

    // start near the "start" wall
    MapStart=Location(startRoom.Walls["start"], ,10);
}
```


Mapper
Zoom: 45.0 %
Zoom
Zoom Fit
Legend
X – Thing
S – MapStart
RUN-TIME CONSOLE


100
25

3. Language Reference Manual

3.1. Language Semantics

A Mapwad program is a standard ASCII text file that can be generated on any computing platform that has an ASCII editor. Mapwad is a case-sensitive language that matches seven types of tokens: white space and comments, identifiers, number, strings, operators, keywords, and other tokens.

3.1.1. White space and Comments

3.1.1.1. White space

Spaces, tabs, newlines, and carriage returns are all considered whitespace. All white space will be ignored.

3.1.1.2. Comments

C and C++ style comments are supported. That is, "//" start a single-line comment, while all text, multi-line or single-line, between "/*" and "*/" will be considered a comment. All text that is considered a comment will be ignored.

3.1.2. Identifiers (Variable, Function)

Identifiers are used for naming variables, functions. They can contain a mixture of alphanumeric characters as well as the underscore character ('_'). It should be noted, that identifiers must begin with either a letter or an underscore. This follows the C specification.

3.1.3. Numbers

Mapwad supports integer and floating point numbers. Integers consist of one or more digits. Floating point numbers can be defined in a couple different ways, with or without a decimal point, exponent, or the combination of the two. An exponent is defined as either the 'e' or 'E' character, followed by an optional sign character ('-' or '+'), followed by an integer. For example:

```
5          // integer or float (depending on type specified at declaration)
0.45       // float
5.2        // float
5e10       // float
5.3E-10    // float
```

3.1.4. Strings

Strings are character sequences enclosed within a set of double quotes. To place a double quote inside the string type two double quotes. The string constant (string of characters enclosed by double quotes) must be contained in one line.

For example, the following assignment,

```
string text = "He said, ""hi there""";
```

would set `text` equal to: He said, “hi there”.

NOTE : C style escape characters such as ‘\n’ or ‘\t’ are not supported in Mapwad string constants.

3.1.5. Null

Mapwad supports a null value like Java.

3.1.6. Operators

=	%	--	%=	<=
+	||	+=	==	.
-	&&	-=	>	
*	!	*=	<	
/	++	/=	>=	

3.1.7. Keywords

The following identifiers are reserved as keywords:

Map	Location	string	while	false
Room	int	if	break	return
Wall	float	else	continue	null
Thing	boolean	for	true	

3.1.8. Other Tokens

The following tokens are used for building statements:

,	(	{	[
;	)	}	]

3.2. Program Layout

A Mapwad program is a combination of variable and function definitions as well as a special “Map” program entry function. It should be noted that any variable or function must be declared before use. Global variables and functions are available in any function, regardless of the where they are declared in the program (like Java).

3.3. Variables

Mapwad supports both explicit/user-defined and implicit variables. User defined variables must be declared before use and have a specific structure for declaration and assignment. The structure forces types on variables at declaration time making Mapwad a strongly-typed language. Implicit variables are Mapwad built-in variables that are associated with specific types.

3.3.1. Explicit (User-Defined) Variables

3.3.1.1. Definition

A user-defined variable is declared in the following manner:

```
<type> <identifier>
```

3.3.1.2. Assignment

As stated above, a variable cannot be assigned a value or accessed until it has been declared. However, Mapwad does allow for variable initialization, that is, a variable can be assigned a value at the time that it is declared.

```
// incorrect – invalid declaration
a;
a = 5;

// correct
int b;
b = 5;

// correct
int c = 5;
```

3.3.1.3. Scope

Variables in Mapwad can have either global or local scope depending on where they are declared. If a variable is declared in the program body (i.e outside function definitions), it is automatically assigned a global scope and can therefore be accessed by any construct that follows in the program (i.e. function definitions, conditionals, loops, etc). Variables that are declared within functions and constructs are local to that function or construct.

It should be noted that Mapwad uses static scoping.

```
int a = 5; // exists within the following if statement
if(a > 3)
{
   int b = 6; // exists only within this if statement
}
```

3.3.2. Implicit Variables

Mapwad includes a list of implicit variables that are associated with some of the built-in types (see Types/Objects section for a list of implicit variables belonging to the different types and objects). Furthermore, an implicit variable may or may not be read-only (see specific type or object for information on its associated implicit variables).

3.3.2.1. ReadWrite/ReadOnly Access Control

Implicit variables can be either ReadWrite or ReadOnly. When a variable for a basic type is ReadOnly the value of the variable cannot be changed. For advanced types (ie objects), ReadOnly means that the object referenced by the variable cannot be changed, although implicit variables can be changed (as long as the implicit variables themselves are not ReadOnly).

For example:

```
Room.Walls = Room2.Walls; // compile error because Walls is
ReadOnly
Room.Walls[4].Name = "Wall 4"; // allowed
```

3.4. Types/Objects

Mapwad has many built in types and objects. The basic types include booleans, integer numbers, floating point numbers, and string literals. The advanced types and objects include Walls, Rooms, Things, Locations, and arrays. It should be noted that Mapwad does not support type casting, but does have limited support for type promotion. If an integer is added to a float, the integer is promoted to a float, and the result is a float.

3.4.1. Basic Types

3.4.1.1. boolean

Boolean values that can take the values 'true' or 'false'.

3.4.1.2. int

A 32-bit signed integer.

3.4.1.3. float

A 32-bit floating point number.

3.4.1.4. string

Basic string of characters enclosed in double quotes. As explained in the first section, if double quotes are needed in the string itself, use a repeated double quote as an escape sequence (see first section for an example). Mapwad's support of strings is very basic and extends only to string creation. Other string manipulation functions (i.e. regular expressions) are not supported.

3.4.2. Advanced Types/Objects

Mapwad has several built-in objects and advanced types. Mapwad's objects are struct-like constructs. Because of their complex nature, each of these objects requires a form of instantiation, and therefore has an associated constructor. Furthermore, as is done in Java, all advanced types and objects are passed by reference.

3.4.2.1. Wall

Walls are used to make up Rooms. Each of the Wall's implicit variables has a default value.

Walls can technically be created outside the context of a Room, but their main purpose is to be connected together to form a Room. Since each Room has a list of Walls (see Room definition), the connection between a Wall and its neighbors is built into the Wall object. That is, each Wall stores the angle between itself and the next Wall in the Room's Walls list.

Furthermore, in order to create a multi-Room environment, there must be a means of interconnecting Rooms. Mapwad provides this ability with a built-in function to merge Walls from different Rooms (see the Attach() function later in the manual).

3.4.2.1.1. Implicit Variables

Unless otherwise specified, each of the Wall's implicit variables is **ReadWrite**.

`string Name` – Name associated with a wall. Default value is "".
`float Length` – The length of the wall. Default value is 20.
`float AngleNext` – The angle between this Wall and the next Wall in a Room's Walls list. Default value is 90.
`string Texture` – The type of texture to be mapped onto the wall (inherited from the quake .WAD file – to be defined later). Default value is "".
`Wall Next` – The next Wall in the Room's Walls list (read-only). Default value is this Wall.
`Wall Prev` – The previous Wall in the Room's Walls list (read-only). Default value is this Wall.
`boolean IsEntry` – Whether or not this Wall is connected to another Wall (read-only). This variable can only be set by a built-in Attach-type function. Default value is false.
`Wall ConnectedTo` – The Wall that this Wall is connected to (read-only). This variable can only be set by a built-in Attach-type function. Default value is null.
`Room FromRoom` – The Room that this Wall is a part of (read-only). Default value is null.

3.4.2.1.2. Wall Constructor

Wall(float Length=20, float AngleNext=90, string Name="", string Texture="")

To create a Wall, use the following syntax:

```
Wall <var name> = Wall(…);
```

3.4.2.2. Room

The Room object is a struct-like construct that is arguably the most important object in Mapwad. Simply stated, a Room is a connection of Walls. A requirement for this connection is that it must be closed (the end point of last wall must be the start point of the first wall). Any Room that is left unclosed will be closed automatically. It should be noted that Walls are allowed to overlap although it is not advisable for logical reasons. Furthermore, Mapwad allows the attaching of rooms in such a way that a courtyard will be created inside (ie a donut). These Walls are stored in a special implicit Walls list that is associated with each Room object.

Central to the Room object is the list of Walls that make up the perimeter. The Walls list is a super data structure that has the dynamic growth functionality of a linked list and the ease of access provided by arrays and associative arrays (where the key is the name of a given Wall). When a Room is first created, it's Walls list is null.

Access to a Room's Wall objects is accomplished by either indexing into the Walls array (i.e. Walls[0], Walls[1], and Walls[2]), or grabbing the first Wall object's handle and iterating using a Wall object's list properties (i.e. Walls[0].Next refers to Walls[1], and so on, with the last Wall's Next variable pointing to Walls[0]). If a Wall in the Walls list has a name, it can be accessed associatively by that name (i.e. Walls["three"]). It should be noted, that if multiple Walls have the same name and associative access is desired, the first Wall in the list relative to Walls[0] that matches will be returned.

Adding and removing Wall objects from the Walls list is done using the built-in functions Add(), AddAfter(), and Remove() (see description of these built-in functions later in the manual).

The flexibility of the Walls list allows the programmer to iterate through a Room's Wall objects in a few different ways (assume x is the name of a Room object):

a) Iterate using normal array functionality
```
Wall currentWall;
int i;
```

```
for(i=0; i<x.Walls.Size; i++)
{
     currentWall = x.Walls[i];
}
```

b) Iterate using list functionality

```
Wall iterator = x.Walls[0];
Wall currentWall = iterator;
iterator = iterator.Next;
for(;iterator!=x.Walls[0]; iterator=iterator.Next)
{
     currentWall = iterator;
}
```

3.4.2.2.1. Implicit Variables

Each of the Room's implicit variables is ReadOnly.

`Wall[] Walls` – List of Walls associated with the Room object (explained above). Walls list is empty (null) by default.

The Walls list itself has an implicit variable:

`int Size` – Number of Walls currently in the list (read-only).

3.4.2.2.2. Room Constructors

Room()

To create a Room, use the following syntax:

```
Room <var name> = Room(..)
```

3.4.2.3. Thing

Things are simple entities supported by Quake that can be put within Rooms. Things include monsters, lava pits, guns, power-ups, etc. A Thing is placed into a Room using a Location (see below).

3.4.2.3.1. Implicit Variables

Each of the Thing's implicit variables is read/write.

`string Type` – Type of Thing. Default value is "".
`Location Position` – Thing's start location. Default value is null.

3.4.2.3.2. Thing Constructor

Thing(string Type="", Location Position=null)

To create a Thing, use the following syntax:

```
Thing <var name> = Thing(..)
```

3.4.2.4. Location

A Location is a three-tuple consisting of a Wall object, a string representation of a position relative to the Wall, and a distance into the Room, measured perpendicular from the Wall.

3.4.2.4.1. Implicit Variables

Each of the Location's implicit variables is read/write.

`Wall NearWall` – The Wall that this Location will be relative to. Default value is null.
`string WallPosition` – Where along the NearWall to place this location. This can take the values, "start", "end", or "center". The default value is "center."
`float WallDistance` – How far out into the Room from NearWall this Location will be positioned. Default value is 5.

3.4.2.4.2. Location Constructor

Location(Wall NearWall=null, string WallPosition="center", float WallDistance=5.0)

To create a Location, use the following syntax:

```
Location <var name> = Location(..)
```

3.4.3. Arrays

Mapwad arrays are very similar to C and Java arrays. Before an array variable can be used, it must be assigned to an allocated array, declared with the number of elements to be allocated. For simplicity, Mapwad also allows the use of array constants, where an array is initialized from a static list of values. Elements in an array are indexed by number starting with 0. The array type extends to every type and object that Mapwad supports. Furthermore, multidimensional arrays are allowed.

3.4.3.1. Usage

The syntax for array variable declarations follows Java except that the brackets are glued to the type and not the variable name. For example to create an array variable,

```
int[] a; // correct
int b[]; // incorrect
```

There are two ways to set the values in an array. First, an array variable may be assigned to an allocated array of the same type. An array is allocated by specifying the array type, followed by the size of the array in square brackets. Second, Mapwad supports array constants where all values are enclosed in curly braces, and separated by commas. Access to elements in the array is performed by using the variable name immediately followed by a left bracket, then the index, then a right bracket. For example, to create an array called numbers that will hold 4 integer values 1,2,3,4, the following two implementations would be correct.

```
int[] numbers = int[4];
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
numbers[3] = 4;
```

or,

```
int[] numbers = {1,2,3,4};
```

Multidimensional arrays are initialized using the following syntax:

```
int[][] numbers = int[2][4];

numbers[0][0] = 1;
numbers[0][1] = 2;
numbers[0][2] = 3;
numbers[0][3] = 4;
numbers[1][0] = 1;
numbers[1][1] = 2;
numbers[1][2] = 3;
numbers[1][3] = 4;
```

Multidimensional array constants are not supported.

3.4.3.2. Implicit Variables

Mapwad arrays have an implicit variable, Size, of type int, that stores the maximum number of elements that the array can contain. The Size variable is ReadOnly.

`int Size` – Maximum number of elements that the array can store. For multidimensional arrays each dimension has a size. So given the numbers array from the previous example, numbers.Size is 2 and numbers[0].Size is 4.

3.5. Statements

Statements make up the heart of a block of code. Every code block is made up of multiple statements that define how the block operates. If a code block is thought of as a recipe, then the variables would be the ingredients and the statements would be the instructions for how to cook the recipe. Within Mapwad there are four basic types of statements. An operator statement (also called an expression), an assignment statement, a control flow statement, and a function call statement.

3.5.1. Operator Statements

3.5.1.1. Arithmetic Operator

Arithmetic operators allow basic arithmetic to be performed on numerical values. Mapwad allows arithmetic operators to be performed on integer variables, float variables, or constant numerical values.

Mapwad supports multiple types of arithmetic operators. The first set are the +, -, *, /, and % operators. These operators take two variables or constants and return the value of the operation. They must be combined with another statement to be useful within the program.

The next set of operators are the assignment arithmetic operators. These are +=, -=, *=, /=, and %=. These operators act similar to the operators above except that the left hand operand must be a variable that can store the result of the operation. So x=x+5 would be written as x+=5.

Finally there are two more arithmetic operators. They are the increment (++) and decrement (--) operators. These operators act similarly to the assignment operators above except that they only have a left hand operand. The right hand operand is assumed to be 1. So x+=1 would be written as x++.

Mapwad supports limited upcasting for arithmetic operations. In any operation that involves a float and an int, the int will be promoted to a float.

3.5.1.2. Boolean Operators

Mapwad supports two different types of Boolean operators. The first type compares two values and returns true or false. These operators are ==, !=, <, >, <=, >=. The == and != are valid for all types, while the other comparison operators are only valid for ints and floats. You can mix ints and floats and Mapwad will upcast an int to a float in order to perform the operation. They are also valid for strings, Booleans, and object variables (for object variables, they test whether the variables refer to the same object). However, both operands must be of the same type and the other operators(<, >, <=, >=) are not supported for strings.

Beyond those operators Mapwad also allows Boolean algebra. Mapwad supports &&, ||, and !. && compares the Boolean value of the left hand operand to the Boolean value of the right hand operand. If both of them are true it returns true, otherwise it returns false. || acts similarly to && except that it returns true if either the left hand or the right hand operand is true. Finally, ! returns the opposite of its right hand operand. ! does not take a left hand operand.

3.5.1.3. Dot Operator

The dot operator '.' accesses a member variable of the object. For example if x is a Wall object then x.Length will access the Length variable associated with x.

3.5.1.4. Precedence

The following chart defines the precedence order for Mapwad. Note the first row has the highest precedence:

f(r,r,..)	(expression)	ID	boolean constant	string literal	int constant	float constant
a[i]						
s.m						
l++	l--					
+a	-a	!a				
a*b	a/b	A%b				
a+b	a-b					
a>b	a<b	a>=b	a<=b			
a==b	a!=b					
X&&y						
x||y						
x=a	x+=b	x-=c	x*=d	x/=e	x%=f	

3.5.2. Assignment

An assignment statement takes a value and assigns it to a specified variable. Within Mapwad only variables can have a value assigned to them. The format of an assignment statement is <variable name> = <value>. The variable name can be any variable that was previously defined and matches the type of the value. Value can be any constant, variable, or statement with a return value. Assuming that `X` and `Y` are integers and `addfunc`() is a function that returns an integer then the following would all be valid assignment statements.

```
X=5;
X=5+1;
X=y-2;
X=addfunc(2,3);
```

3.5.3. Control Flow

A control flow statement describes what statements will be executed next. There are two basic types of control flow constructs, conditional statements and loops. Within all of the control flow statements a section of code is designated with { }. The code within that section has its own scope and any variable that is declared within that section will not exist outside of the section.

3.5.3.1. Conditional Statements

Conditional statements allow for statements to be executed only in specific cases. The format for a conditional statement is as follows:

```
if (<Boolean expression>)
{
   <statement>*
}

 or

if (<Boolean expression>)
{
   <statement>*
}
else if (<Boolean expression>)
{
   <statement>*
}
else
{
   <statement>*
}
```

The statement within the {} for if will be executed if the Boolean expression evaluates to true. If the statement evaluates to false then the next else if statement will be checked and the program will continue to check the else if statements until a Boolean expression evaluates to true. If the program reaches an else without an if those statements are evaluated. Once one Boolean expression returns true and the corresponding block is executed the rest of the conditions are skipped and the program continues after the else or the final else if.

3.5.3.2. Loop Statement

A loop statement executes a block of code repeatedly until a condition is met. There are two types of loops supported in Mapwad, **for** loops and **while** loops. The **break** and **continue** statements work as in Java within the loops.

3.5.3.2.1. For Loop

A for loop in Mapwad follows the same basic syntax as a C/C++/Java for loop.

```
for (<initialization>;<boolean expression>;
      <iteration action>)
{
      <statement>*
}
```

The initialization can be any valid expression; however it is intended to initialize a variable for the loop. The one exception to this is that a variable can not be declared in the initialization block of **for** statement.

The boolean expression is evaluated each time the loop is iterated. If the value returns true the loop statements are executed one more time.

The iteration action can be any valid expression; however it is intended to increment the loop variable.

3.5.3.2.2. While Loop

The **while** loop is a more general purpose loop than the **for** loop. The syntax for the **while** loop is as follows:

```
while (<Boolean expression>)
{
      <statement>*
}
```

Any valid boolean expression can be used and any number of statements can then be put within the body of the loop. The boolean expression is evaluated before executing the code block on each iteration. If the boolean expression becomes false the loop terminates.

3.6. Functions

A function is a block of code that can be called from anywhere in the program. A function in Mapwad performs similarly to how a function in C/C++/Java performs.

3.6.1. Structure

A function is made up of four parts, a return type; an identifier; an argument list; and the function body. A function definition has the following syntax:

```
<return type>? <identifier>(<argument list>)
{
  <body>
}
```

The return type can be any valid type such as, a basic type, an advanced type. However, if the function does not return a value then a return type is not specified.

The identifier is any valid Mapwad identifier as defined earlier in this manual.

The argument list is a list of 0 or more variable declarations (type and name) separated by commas. Any variable can have a default value specified. If a default value is specified then the variable is optional when the function is invoked. A parameter list would look like the following:

```
(int x, int y=1, int z=2, string a="default", string b)
```

In the above example, variables `y`, `z`, and `a` are all optional variables that do not have to be specified when the function is called.

The body of the function is made up of 0 or more statements.

3.6.2. Invocation

A function is invoked in a statement by its name followed by the required arguments enclosed in parentheses. The following example would call a function called myFunc with the parameter list defined in the previous section.

```
myFunc(3, ,5 , ,"the end")
```

Recall, that variables `x` and `b` were not optional and it is therefore required that values be included in the argument list. Variables `y`, `z`, and `a` were optional and could be left blank. However, `z` was included in the parameter list so that the new value of `z` will be used within the function instead of the default value of 2. Notice that any value that was not included in the function invocation still included the comma separator to indicate that the default should be used. However, if the parameter list includes optional parameters at the end of the list then the function can be called without including the extra commas. So, given a function myFunc2(int x=1, string y="name", int z=3) all of the following would be valid function calls:

```
myFunc2(5,"hi",9) //x=5, y="hi", z=9
myFunc2(6)        //x=6, y="name", z=3
myFunc2()         //x=1, y="name", z=3
```

3.6.3. Returning

If a function has a return type it must have a return statement somewhere in the body of the function. When the program flow reaches a return statement, the value given in the return statement is returned to at the point in the program from which the function was called, and the function terminates.

The function invocation can be a statement by itself or it can be part of a larger statement. If the function invocation is a statement by itself then any return value is ignored. However if the function invocation is included as part of an expression then it must return a value. Once that value is returned, the rest of the expression executes as if that value had been supplied instead of the function call.

3.6.4. Map Function

Mapwad has one special function, the Map function. This is equivalent to the main function in C. The Map function is run first and is used to describe the layout of the entire map.

In addition, there are a few implicit global variables that should be assigned somewhere in the program.

`MapStart`	Location Defines where the player starts when the final map is run in the game engine. MUST be defined in order for the program to compile.
`MapHeight`	float Defines the height of each room. Default value - 10
`FloorTexture`	string Defines the texture of the floor. Default value - “GROUND1_5”
`CeilingTexture`	string Defines the texture of the ceiling. Default value - “GROUND1_5”.

3.6.5. Built-in Mapwad Functions

Mapwad includes some predefined functions. They are defined as follows:

```
boolean Add(Room room, Wall newWall)
```

This function adds `newWall` to the end of the Walls linked list associated with `room`. Add() will return true if the operation succeeds. If you attempt to add a Wall that already exists in a Room other than `room`, then Add() will return **false**. When called on a Room with an empty Walls list (i.e. a Room that was just initialized), Add() will initialize the list, setting `room.Walls[0]` to `newWall`.

```
boolean AddAfter(Wall locat, Wall newWall)
```

This function adds `newWall` to the linked list of Walls that contains `locat`. `newWall` is added after `locat` in the linked list. `AddAfter()` will return **true** if the operation succeeds. If you attempt to add a Wall that already exists in a Room other than `room`, then `AddAfter()` will terminate and return **false**.

```
Remove(Wall rem)
```

Remove will remove the Wall `rem` from the Walls linked list of the Room that contains `rem`.

```
boolean Attach(Wall a, Wall b, string wallconnect)
```

This function is used to connect two rooms at the point specified by Wall `a` and `b` within each room.

Walls `a` and `b` are walls in separate rooms that are going to be connected with an entry. `Attach()` connects `a` to `b` and to remove their intersection. `wallconnect` indicates where the two walls will be connected relative to each other, and can take the values "start", "end", or "center". That is, "start" connects the wall `a`'s start point with wall `b`'s end point, "end" connects wall `a`'s end point with wall `b`'s start point, and "center" connects wall `a`'s center point to wall `b`'s center point. `Attach()` will return **true** if it succeeds and **false** if it fails.

```
boolean IsClosed(Room rm)
```

This function checks if the room is closed. A room is considered closed if the final wall in the room ends at the same location as the start of the first wall. True is returned if the room is considered closed and false is returned otherwise.

```
boolean Close(Room rm)
```

This function first checks if the room is closed. If the room is closed the function returns true. If the function is not closed then a new wall is created that goes from the end of the last wall back to the start of the first wall. This closes the room and true is returned. Note that this wall takes the shortest path back to the start wall and will possibly cross other walls to get there. False is only returned if there is only one wall and it is impossible to close the room.

```
boolean IsPath(Room|Wall|Location rm1, Room|Wall|Location rm2)
```

IsPath() checks to see if it is possible to walk from rm1 to rm2. If there is a useable path from rm1 to rm2 then true is returned otherwise false is returned.

```
boolean Print(variable var)
```

Prints the value of `var` on the standard output. This is the only built-in function that can take any Mapwad type as an argument. For basic type variables, `Print` will print the value of the variable. For advanced type variables, `Print` will print the identifier of the Java instance of the advanced type, which can be useful for identifying if two variables refer to the same object. `Print` is intended for debugging and status display. `Print` was used to test the Mapwad compiler.

4. Project Plan

4.1. Planning

Mapwad's general design and specifications were outlined during frequent group meetings. At the outset of the project, our meetings focused around defining our goals for a general language and then ultimately for the Mapwad programming language. As the project progressed, we began designing and specifying the different components and constructs that would actualize our goals. As development began we continued to meet and refine Mapwad's design to account for certain implementation pitfalls. The constant communication within our group was central to Mapwad's success, as we were able to keep each other focused and involved, and most importantly, we were always on the same page (or very close to it).

4.2. Development

Mapwad's code development was separated into sections, or modules, and divided among the group members. These modules included:

- Pre-processing
- Internal Processing
- Post-Processing

In order that there would be no breakdowns in communication (which is almost inevitable when a project is broken up into independent pieces), we tried to do develop at the same time and in the same place. For the most part we were able to do this, and as such, any problems that arose were resolved almost immediately. We used Java's Javadoc tool extensively to document our code. This gave each of us an easy way of knowing what code was written (and not written) and how the code was meant to be used. The generated documentation was updated regularly and posted to the web.

4.3. Testing

Testing was performed both during module development and during module integration. For each individual module, code samples were written to test the functionality. These code samples evolved with the modules, and therefore provided regression testing for the module. When it came time to integrate the modules we actually used test code that was written before development began. Aside from just being test code, these code samples allowed us to see how well we "stayed on course" with regard to our initial goals. In addition to these code samples, we adapted some code used for module testing (basic regression testing).

4.4. Programming Style

Since each of us had our own coding style, we did not define strict coding rules. We did however, require that the code be readable and also that it be commented in a way that any of us could read and understand it. Furthermore, we did our best to write basic javadoc style comments for functions and classes.

The following is a list of suggested stylistic rules outlined before development:

- Javadoc
    - Document all methods public and private.
    - Document all Classes.

```
/**
 * ShortDescription.
 *
 * Long description.
 *
 * @author (for classes)
 * @version CVS $Id$ ($Id$ will be expanded by CVS - for classes)
 * @param type varName Description.
 * @return type Description.
 * @throws ExceptionName Description.
 */
```

- Standard Java style naming
    - Methods and Normal variables: normal java style capitalization.
        - numRooms
    - Final variables: all caps underscore between logical words.
        - NUM_ROOMS
    - Object member variables: m_ precedes variable name.
        - m_VarName
        - This actually turned out to be very useful as member variables corresponded to an Object's implicit variables (which in Mapwad are capitalized).

- Control Flow:
    - Indentation is four spaces (DO NOT indent curly braces into block)
    - A one line conditional or loop does not require braces.
    - Braces for block statements on separate line.

        -
          ```
          if()
          {
          }
          ```

CVS was used for version control, and Java packaging was used to create a logical separation between modules. In the end we found that we were very successful in keeping our code easy to understand, as each group member at some time or another ended up modifying or adding other modules without help from the author.

4.5. Tools

There are many tools that were employed in the development of Mapwad. Development was done on CS machines (clic or cluster) in the Emacs editor. Java was the programming language of choice for all of the Mapwad Compiler's internals. ANTLR was used to generate the necessary Java source code for the lexer, parser, AST, and walker. We exploited Java's Javadoc code documenter to keep an up to date record of what was coded and what the code did. When necessary, JSwat was used for debugging. As mentioned above, CVS was used for version control. It should be noted that all Java code was compiled with the Java version 1.4.1.

Once compiled, a Mapwad program is sent through qbsp, an open source Quake compiler, that creates a binary file understandable by Quake. And of course, Quake is needed to experience the power of Mapwad.

4.6. Timeline and Project Log

Proposed Timeline

Goal	**Due Date**
Initial design meetings	September 2003
White paper	September 25, 2003
Design and specification meetings	October 2003
Initial release of lexer/parser	Late October 2003
Language reference manual	October 28, 2003
Module assignments	Early November 2003
Initial Module Release	Mid/Late November 2003
Initial Integration	Late November 2003
Final Module Release/Integration	Early December
Final Testing	Early/Mid December 2003
Final Documentation	Mid December 2003

Actual Timeline (dates based on CVS logs)

Goal	**Due Date**
Initial design meetings	September 2003
White paper	September 25, 2003
Design and specification meetings	October 2003
Language reference manual	October 28, 2003
Initial release of lexer/parser	October 28, 2003
Module assignments	Early November 2003
Initial release of modules	December 2, 2003
Initial integration	December 2, 2003
Final release of modules	December 8, 2003
Final integration (another final release of modules!)	December 16, 2003
Final testing/sample program generation	December 16-17, 2003
Final documentation writing/editing	December 17-18, 2003

4.7. Group Member Responsibilities

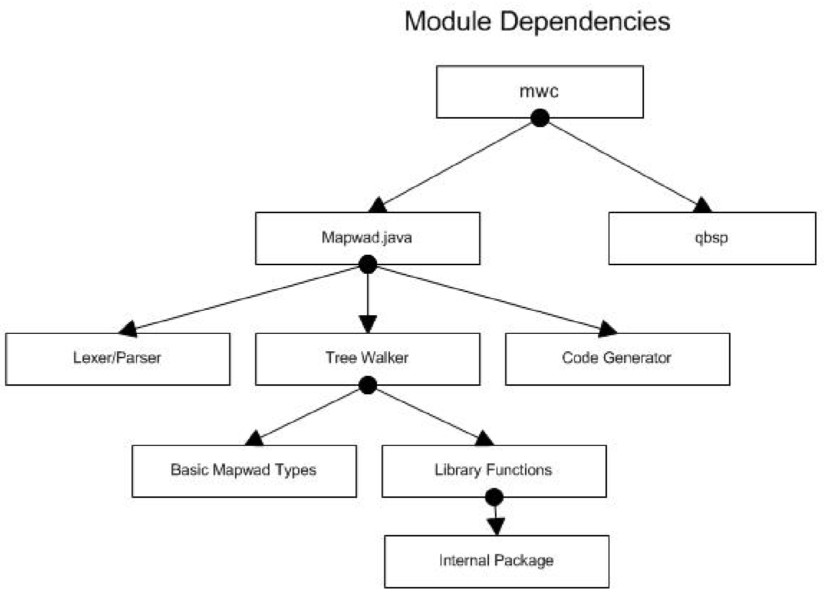
Group Member	**Responsibilities**
Ben Smith	Lexer/Parser, Walker, Semantic Checker, Testing
Avrum Tilman	LRM, Internal Package, Library Functions, Testing

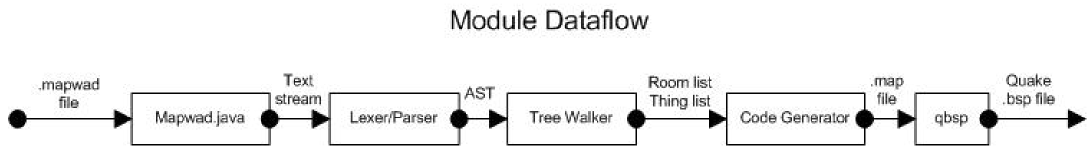
Josh Weinberg	LRM, Internal Package, GUI, Testing
Ron Weiss	Lexer/Parser, Internal Package, Post-processing, Library Functions, Testing

The white paper, presentation, and final report were done as a group.

5. Architectural Design

5.1. Block Diagrams





5.2. Architecture Description

5.2.1. Mapwad Compiler Components

mwc

The Mapwad compiler is initiated using a shell script named mwc (**M**ap**w**ad **C**ompiler). It runs Mapwad.java to convert Mapwad code into .map intermediate text format, and then runs qbsp to convert the .map file into a .bsp Quake level file that the rendering engine can understand. The following is an explanation of what the different compiler pieces do.

1. Mapwad.java
   Translates Mapwad code into an intermediate .map text format for use with qbsp. Contains the main() method from which it all happens. The components of the compiler are as follows:
   a. Lexer/Parser
      Written in Antlr. The lexer converts the Mapwad code (written in text format) into a stream of tokens that the parser then converts into an abstract syntax tree.
   b. Tree Walker

This is the heart of our compiler. The tree walker takes the AST created by the parser and walks through it. It is a two-pass compiler that first gathers information about global function and variable definitions and then traverses the tree a second time to check types, resolve symbols, and execute code.

1. Basic Mapwad Types:
   Java classes represent the basic built-in types in Mapwad and include the code to manipulate them.
2. Library Functions:
   These functions are wrappers for the various built-in library functions in Mapwad. They include math functions, print functions, Room manipulation functions, as well as constructor wrappers for the different object types supported by Mapwad. These functions serve as the interface between the Mapwad code executed by the tree walker and the internal data structures used to keep track of Rooms as they are attached.
3. Internal Package
   This package manages internal data structures representing the different object types, including Rooms, Walls, Things, and Locations. It includes code to manipulate and assign coordinates to the Walls located in each Room, which involves keeping track of the locations of Rooms as they are attached to each other (through various geometrical operations: translation and rotation of Rooms to ensure that attached walls line up with each other). The code in this package is called by the library functions (discussed above) during the traversal of the AST.

c. Code Generator
   Called by Mapwad.java after the AST has been traversed. The generator takes a list of Things and a list Rooms created during the execution of the Map() function in the Mapwad code. It then assigns global coordinates to these Rooms and Things and converts them to entries in the Quake .map file. Each Wall in the map is translated into a 'brush' consisting of the intersection of 6 planes in space. Each Thing is converted into an 'entity' statement consisting of a Thing name and the coordinates within the map. This generator is what actually generates the actual output .map file for use with qbsp.

2. qbsp
   Translates the .map file created by the Mapwad compiler into a Quake 1 compatible .bsp map file that can be loaded from the game.

5.2.2. Mapwad Compiler Components by File

mwc (Ron)

Mapwad.java (Ben) - responsible for the compilation of .mapwad code into .map intermediate text format. It Calls the following:

Lexer/Parser
- grammer.g (Ron/Ben)

Tree Walker (Ben)
- walker.g
- MWAST.java

Basic Mapwad types: (Ben)
- internal/MWArray.java
- internal/MWBoolean.java
- internal/MWDataType.java
- internal/MWFloat.java
- internal/MWInt.java
- internal/MWReferenceType.java
- internal/MWString.java

Library Functions (Avrum/Josh/Ben/Ron)
- -Object constructors:
  - MWLocationConstruct.java
  - MWRoomConstruct.java
  - MWThingConstruct.java
  - MWWallConstruct.java
- -Math functions:
  - MWMath.java
- -Room/Wall functions:
  - MWAdd.java
  - MWAttach.java
  - MWIsClosed.java
  - MWIsPath.java
  - MWRemove.java
- -Misc Library Functions:
  - MWPrint.java
  - MWRandInt.java
  - MWRandFloat.java

Internal Package/Object types (Avrum/Josh)
- internal/MWLocation.java
- internal/MWRoom.java

internal/MWThing.java
internal/MWTransform.java
internal/MWWall.java
internal/MWWalls.java

Code Generator
internal/MWGui.java (Josh)
internal/MWMap.java (Ron/Josh/Avrum)

qbsp (id Software, slight modifications by Ben/Ron)

6. Test Plan

As is the case with all compilers, it is critical that the Mapwad compiler (MWC) function as specified in the Language Reference Manual. Given a Mapwad program, the MWC should interpret the instructions in that program according to the language specification, and the MWC should halt with an appropriate error message when the program does not conform to the language specification. It would, of course, be very difficult to write Mapwad programs if the MWC were to interpret programs in a manner inconsistent with the specifications, or if the MWC ignored or misinterpreted errors in the program.

6.1. Approach to Testing

The MWC is organized as modules, each of which has a specific role in the compilation of a Mapwad program. It was therefore natural to test each module independently, as development progressed, until the modules were ready to be integrated. Once the modules were finished and integrated, testing of the entire compiler began.

6.2. Testing the Parser

Performed by Ben Smith and Ron Weiss.

Testing of the parser began even before all the details of the language had been worked out. Testing consisted of running the parser against Mapwad programs written to conform to the language as specified at that time, and checking the result. The Mapwad program parsertest.mapwad produces a single room, and is not a very interesting example of what the language can do. However, the program uses most of the syntactic elements of the language.

```
/*
 * parsertest.mapwad-attempts to test all of the features of our
 * MAPWAD parser
 */

Map()
{
      bool = false;

      Room r = BigRoom(,100);
      Location l = Location(r.Walls[s], "center", 1);

      MapStart = l;

      // stupid boring while loop to count number of walls in r
      Wall w = r.Walls[0];
      Integer = 1;
      while(w.Next != r.Walls[0])
      {
            Integer++;
            w = w.Next;
      }
```

```
        Print(Integer);
        Print(r.Walls.Size);

        //some worthless math calculations
        f *= ((f+2)/3 + 1e44/Integer)*add();

        //func calls with optional arguments
        add(1,);
        add(,);
        add();
        add(,122);
        add(2,23.3);
}

//define some global variables
int Integer;
float f = 1.23423e-9;
boolean bool = false;
string s = "Hello my name is Simon and I like to do drawings.";

string getString(string[] strings, int x=0)
{
        return strings[x];
}

float add(float a = 1, float b = 2)
{
        return a+b;
}

Room BigRoom(int num = 4, int walllength)
{
        int n;
        int numwalls;

        if(bool)
        {
                n = 4;
        }
        else if(!bool)
        {
                n = num;
        }

        if(n < 3)
        {
                numwalls = 3;
        }
        else
        {
                numwalls = n;
        }

        int wallangle = (numwalls-2) * 180;

        Room r = Room();
        int x;
        for(x = 0; x<numwalls; x+=1)
        {
              Add(r,Wall(walllength, wallangle));
        }

        r.Walls[0].Name = s;

        int perimiter = numwalls*walllength;

        return r;
}
```

parsertest.mapwad was a useful testing tool. After the parser reached the point where it could successfully parse parsertest.mapwad, it did not need to modified again because of syntactic problems. The only changes to the parser were to modify the AST tree so as to resolve semantic problems in the walker. As development progressed, we continued to use parsertest.mapwad to verify that changes did not affect the parser.

6.3. Testing the Walker

Performed by Ben Smith.

During early development of the walker, testing was done using small Mapwad programs written to test one or two features. For example, a Mapwad program was written containing a user defined function along with several calls to that function with different parameters to test that the walker could handle user defined functions correctly. As the walker become more sophisticated, it became possible to test it on sample Mapwad programs (see the Mapwad tutorial...) we had written for the LRM. These turned out to be extremely useful, as the programs were intended to show off the features of the language, and therefore were not written with the same bias of programs that are written specifically to test the walker. paresertest.mapwad also turned out to be useful in testing the walker, since in addition to using nearly all the syntactic elements, it also tests much of the semantics.

Some elements of the language were tested using Mapwad programs written to produce an expected output. The mathematical operators, for example, were tested with the following program, optest.mapwad.

```
Map()
{
      Print("Int ops");
      TestIntOps();
      Print("");
      Print("Float ops");
      TestFloatOps();
      Print("");
      Print("Mixed ops 1");
      Print("");
```

```
        TestMixedOps1();
        Print("");
        Print("Mixed ops 2");
        TestMixedOps2();

        Print("");
        Print("Boolean ops");
        TestBoolOps();
}

TestIntOps()
{
        int test1=4;
        int test2=2;

        Print(test1+test2);
        Print(test1-test2);
        Print(test1*test2);
        Print(test1/test2);
        Print(test1%test2);

        test1 += test2;
        Print(test1);

        test1 -= test2;
        Print(test1);

        test1 *= test2;
        Print(test1);

        test1 /= test2;
        Print(test1);

        test1 %= test2;
        Print(test1);

        test1++;
        Print(test1);
        test1--;
        Print(test1);
}

TestFloatOps()
{
        float test1=4;
        float test2=2;

        Print(test1+test2);
        Print(test1-test2);
        Print(test1*test2);
        Print(test1/test2);
        Print(test1%test2);

        test1 += test2;
        Print(test1);

        test1 -= test2;
```

```
      Print(test1);

      test1 *= test2;
      Print(test1);

      test1 /= test2;
      Print(test1);

      test1 %= test2;
      Print(test1);

      test1++;
      Print(test1);
      test1--;
      Print(test1);
}

TestMixedOps1()
{
      int test1=4;
      float test2=2;

      Print(test1+test2);
      Print(test1-test2);
      Print(test1*test2);
      Print(test1/test2);
      Print(test1%test2);

      test1 += test2;
      Print(test1);

      test1 -= test2;
      Print(test1);

      test1 *= test2;
      Print(test1);

      test1 /= test2;
      Print(test1);

      test1 %= test2;
      Print(test1);
}

TestMixedOps2()
{
      float test1=4;
      int test2=2;

      Print(test1+test2);
      Print(test1-test2);
      Print(test1*test2);
      Print(test1/test2);
      Print(test1%test2);

      test1 += test2;
      Print(test1);
      test1 -= test2;
      Print(test1);

      test1 *= test2;
      Print(test1);

      test1 /= test2;
      Print(test1);

      test1 %= test2;
      Print(test1);
}

TestBoolOps()
{
      boolean test1=true;
      boolean test2=false;

      Print(test1 && test2);
      Print(test1 || test2);
      Print(!test1);
}
```

The output of optest.mapwad can easily be used to verify that the mathematical operators in Mapwad function as described in the LRM.

6.4. Testing the Internals

Performed by Avrum Tilman and Josh Weinberg.

We needed to be able to test the internal module before the input module was ready to be integrated with it. Since the input module was not connected, Mapwad programs could not be used as test cases. Instead we wrote our own test code in java that called our objects and functions to test the functionality. We continued to build on the same basic test file adding more features as we completed larger parts of the language. Here is the final test program that we used:

```
import internal.*;
import java.util.*;

public class MapTester
{
    public static void main(String args[]) throws Exception
    {
      Vector rooms = new Vector();
      MWRoom rm1 = new MWRoom();
      MWRoom rm2 = new MWRoom();
      MWRoom rm3 = new MWRoom();
      MWRoom rm4 = new MWRoom();
      MWRoom rm5 = new MWRoom();
      MWRoom rm6 = new MWRoom();
      MWWalls ws1 = rm1.getWalls();
      MWWalls ws2 = rm2.getWalls();
      MWWalls ws3 = rm3.getWalls();
      MWWalls ws4 = rm4.getWalls();
      MWWalls ws5 = rm5.getWalls();
      MWWalls ws6 = rm6.getWalls();

      ws1.add(new MWWall(new MWFloat(100), new MWFloat(90), new
MWString(), new MWString()));
      ws1.add(new MWWall(new MWFloat(150), new MWFloat(180), new
MWString(), new MWString()));
      ws1.add(new MWWall(new MWFloat(150), new MWFloat(90), new
MWString(), new MWString()));
      ws1.add(new MWWall(new MWFloat(100), new MWFloat(90), new
MWString(), new MWString()));
      ws1.add(new MWWall(new MWFloat(300), new MWFloat(90), new
MWString(), new MWString()));

      ws2.add(new MWWall(new MWFloat(50), new MWFloat(90), new
MWString(), new MWString()));
      ws2.add(new MWWall(new MWFloat(100), new MWFloat(90), new
MWString(), new MWString()));
      ws2.add(new MWWall(new MWFloat(50), new MWFloat(90), new
MWString(), new MWString()));
      ws2.add(new MWWall(new MWFloat(100), new MWFloat(90), new
MWString(), new MWString()));

      ws3.add(new MWWall(new MWFloat(100), new MWFloat(90), new
MWString(), new MWString()));
      ws3.add(new MWWall(new MWFloat(300), new MWFloat(90), new
MWString(), new MWString()));
      ws3.add(new MWWall(new MWFloat(100), new MWFloat(90), new
MWString(), new MWString()));
      ws3.add(new MWWall(new MWFloat(150), new MWFloat(180), new
MWString(), new MWString()));
      ws3.add(new MWWall(new MWFloat(150), new MWFloat(90), new
MWString(), new MWString()));

      ws4.add(new MWWall(new MWFloat(50), new MWFloat(90), new
MWString(), new MWString()));
      ws4.add(new MWWall(new MWFloat(100), new MWFloat(90), new
MWString(), new MWString()));
      ws4.add(new MWWall(new MWFloat(50), new MWFloat(90), new
MWString(), new MWString()));

      ws5.add(new MWWall(new MWFloat(70), new MWFloat(90), new
MWString(), new MWString()));
      ws5.add(new MWWall(new MWFloat(50), new MWFloat(90), new
MWString(), new MWString()));
      ws5.add(new MWWall(new MWFloat(70), new MWFloat(90), new
MWString(), new MWString()));

      ws6.add(new MWWall(new MWFloat(380), new MWFloat(90), new
MWString(), new MWString()));
      ws6.add(new MWWall(new MWFloat(270), new MWFloat(90), new
MWString(), new MWString()));
```

```
      ws6.add(new MWWall(new MWFloat(40), new MWFloat(90), new
MWString(), new MWString()));
      ws6.add(new MWWall(new MWFloat(50), new MWFloat(90), new
MWString(), new MWString()));
      ws6.add(new MWWall(new MWFloat(20), new MWFloat(270), new
MWString(), new MWString()));
      ws6.add(new MWWall(new MWFloat(200), new MWFloat(270), new
MWString(), new MWString()));
      ws6.add(new MWWall(new MWFloat(340), new MWFloat(270), new
MWString(), new MWString()));
      ws6.add(new MWWall(new MWFloat(200), new MWFloat(270), new
MWString(), new MWString()));
      ws6.add(new MWWall(new MWFloat(20), new MWFloat(90), new
MWString(), new MWString()));
      ws6.add(new MWWall(new MWFloat(50), new MWFloat(90), new
MWString(), new MWString()));
      ws6.add(new MWWall(new MWFloat(40), new MWFloat(90), new
MWString(), new MWString()));
      ws6.add(new MWWall(new MWFloat(270), new MWFloat(90), new
MWString(), new MWString()));


      MWString c = new MWString("c");
      c.setValue("center");
      MWString st = new MWString("st");
      st.setValue("start");
      MWString e = new MWString("e");
      e.setValue("end");
      ws1.getWall(1).attach(ws2.getWall(0), st);
      ws2.getWall(2).attach(ws3.getWall(4), st);
      ws3.getWall(3).attach(ws4.getWall(2), st);
      ws4.getWall(0).attach(ws1.getWall(2), st);
      ws5.getWall(0).attach(ws2.getWall(3), c);
      ws6.getWall(3).attach(ws3.getWall(1), e);
      ws6.getWall(9).attach(ws1.getWall(4), st);

      rooms.add(rm1);
      rooms.add(rm2);
      rooms.add(rm3);
      rooms.add(rm4);
      rooms.add(rm5);
      rooms.add(rm6);

      MWString s = new MWString("start");
      s.setValue("center");
      MWLocation start = new
MWLocation(rm1.getWalls().getWall(0), s, new MWFloat(200));

      MWString t = new MWString("thingy");
      t.setValue("center");
      MWLocation thingy = new
MWLocation(rm1.getWalls().getWall(0), t,new MWFloat(70));
      MWString type = new MWString("thingo");
      type.setValue("monster_army");
      MWThing thing = new MWThing(type,thingy);

      Vector things = new Vector();
      things.add(thing);

      MWString t2 = new MWString("thingy");
      t2.setValue("center");
      MWLocation thingy2 = new
MWLocation(rm2.getWalls().getWall(1), t2, new MWFloat(20));
      MWString type2 = new MWString("thingo");
      type2.setValue("item_rockets");
      MWThing thing2 = new MWThing(type2,thingy2);

      things.add(thing2);

      MWString str = new MWString("str");
      str.setValue("");
      MWMap m = new MWMap(rooms, things, start);
      MWGui gui = new MWGui(m);

      m.createMapFile("test2.map");
    }
}
```

Initially we just relied on examining the data structures to confirm that our code was executed correctly. Once the MWGui was completed we were able to examine a graphical representation of the map that was being created. Finally, when the createMapFile function was completed we were able to check see code fully compiled without using the Mapwad language and the lexer/parser/tree walker. By this point we were confident that our code would work when it was integrated in with the rest of the project.

6.5. Testing Code Generation

Performed by Ron Weiss.

We needed to be able to test the output module before the input module was ready to be integrated with it. Since the input module was not connected, Mapwad programs could not be used as test cases. Instead Java code was written utilizing the internal data structures representing Walls, Rooms, and Things which then called the code generation functions and generated an output file. Initial testing consisted only of attempting to generate a single wall that qbsp could compile and Quake could render properly. As the code progressed Java code was written to create more complex maps containing various Things and complex arrangements of Walls. Eventually common test cases were created to verify the correct operation of both the internal package and the code generation portions of the Mapwad compiler. Before the completion of the graphical map preview code, testing the code generation required running qbsp on the generated .map file and loading it in to Quake to ensure that it created a map that looked like what was expected. This process was quite tedious and thus progressed rather slowly once the basic code generation of walls was working. Once MWGui was written and we were able to see a two dimensional representation of a map before viewing it in 3D in Quake, it became possible to create increasingly complex maps and verify almost immediately in Quake that the code generation was generating output that was equivalent to the internal Mapwad representation.

6.6. Testing the Integrated Compiler

Performed by Ben Smith, Avrum Tilman, Josh Weinberg and Ron Weiss.

Typical regression testing involves designing test cases consisting of a specific input and an expected output, and using these test cases to verify correct behavior of the software throughout the development process. Due to the nature of the Mapwad language, this type of regression testing was not possible.

A typical Mapwad program does not produce an output that can be automatically compared against an expected output so as to verify that a new version of the compiler produces output equivalent to previous versions. Granted, the MWC does produce an output map file, which could be compared against the expected output map; however, the two maps may be equivalent, even if the outputted .map files are not identical. The LRM specifies only that the MWC will generate coordinates with relative positions consistent with the layout specified by the Mapwad program. Selection of the absolute coordinate system outputted by MWC, is non-deterministic. A modification to an algorithm in the internal module might produce coordinates for a map different from those produced by a previous version of the compiler. The map, however, might still be equivalent to the previously generated map in terms of the lengths of the walls and the interconnections between rooms.

Therefore, regression testing of the MWC required a human verification step. We tested the MWC against a set of sample Mapwad programs throughout development, initially verifying that the compiler could parse them, then understand them, and finally produce a map consistent with what we had specified should be produced in the LRM. As we integrated the modules together, we continued to compile these sample programs, verifying that the compiler both accepted the programs and produced correct output.

7. Lessons Learned

7.1. Ben Smith

When writing a language compiler it is very important to generalize language constructs as much as possible. For example, it turned out to be extremely useful to have almost all functions called by the Walker take the MWDataType (the base class for all data types) as input. Later when it became clear that additional layers of indirection where necessary to make object types work, it was relatively straightforward to wrap the existing object classes in a new MWReferenceType derived from MWDataType. Since the language code had been written to work with MWDataTypes, most of it did not need to be modified to accept the MWReferenceType.

The easiest implementation is usually best. When implementing multidimensional arrays, I initially tried to create an array class that would store and manage the entire multidimensional array. First of all this turned out to be somewhat difficult to implement correctly. Then I discovered that although I was allocating memory for the arrays correctly, performing operations on elements of the array was quite difficult because the all inclusive array I had implemented would not work with the walker. I realized that a more simple array implementation, in which each instance of the array class represents a single dimension of the array would work be much easier to integrate into the walker. With MWArray implemented this way, a multidimensional array could be generated by recursively adding MWArrays to MWArrays.

7.2. Avrum Tilman

The most important lesson to be taken out of this project is to understand everything as fully as possible before getting started. This became apparent in two different ways. The first place where this was important is in understanding the full scope of a project. When designing a new language it is very easy to get caught up in the cool features that your language will support. However in most of those cases the cooler the feature the harder it is to implement. Some of the cool features might be important enough that they are worth the time to implement, but in a lot of cases there is not enough time in a one semester project to implement all the ideas you can think of, so the sooner you understand the scope of your ideas the sooner you can eliminate the extra goodies and focus on getting the heart of the project working before wasting time on superfluous features.

The second place where understanding is important is in defining the interfaces between different components. All the interfaces need to be well designed before anyone starts coding or else you will be forced to modify the code when the components are integrated. In addition, beyond just defining the interfaces make sure you understand all the ramifications of the interface design. In our case even after fully specifying the interfaces early we discovered later that our interfaces did not allow the full functionality that we needed so we had to change them.

Also, as in any group project, communication is critical. All group members should know what everyone else is doing at all times. This ensures that no two people end up working on the same issue from different sides of the problem. Also, the more communication between group members the quicker problems can be resolved as one person finds problems/bugs/quirks in another person's code.

Finally, make sure to generalize your code. The more general your code is the easier it is to modify it for new situations that inevitably come up.

7.3. Josh Weinberg

While learning about grammars in Computability last semester, I thought to myself, why the heck are these important? As you can tell, I had no idea what a compiler was all about…truth be told, I had never really given thought to how they were implemented and how they worked! Well, PLT cleared up these issues fast. The exposure to knew computer science concepts like AST's and scoping rules, as well as the idea that some of the theory we have learned is not actually useless, was really very gratifying.

In terms of the group project, four things really stand out as being critical for success: definition of goals early (and sticking to them), communication, communication, and communication! I think that Mapwad was so successful because we knew what we wanted, but we listened to others. From the outset, we made it our business to meet on a regular basis, and keep each other informed of progress and ideas. We sat and designed the language together, all the while making sure that everyone had a say in everything. Before development began we made sure to outline the goals of the Mapwad, that it should be a cool, powerful language that was easy to program. When the development began we kept the lines of communication open by working at the same time and the same place (at least we tried to!). When issues arose regarding the interfacing between modules, we were able to resolve them quickly and efficiently, making sure that Mapwad's goals were still being fulfilled and that everyone was on the same page.

My suggestions for future groups is to start early, define goals (and be prepared to stick with them), and keep the lines of communication as open as possible!

All in all, I was so happy with all our work. My group members did a fantastic job, and we have a really great language to show for it. As projects like these usually push people apart, I think Mapwad was an experience that brought us together!

7.4. Ron Weiss

The absolute most important thing to learn from a group project can be summed up in two words. Start early. The earlier you start, the easier it will be in the end. Fortunately we began working on our implementation immediately after the LRM was due. We had everything integrated and working reasonably well before classes ended. Given the number of other deadlines I had at the end of the semester I am quite grateful for this. One of the reasons I think we were able to do this is because we picked a project that is a lot fun to work on. There's nothing quite like venting your frustrations by playing Quake and still being able to tell people that you're working on a school project. Playing Quake for school. If that's not motivation enough to work on a project then I don't know what is.

A well thought out design is a key component to a successful project. We settled on a basic design early on in the semester and deviated very little from it in the final product. Extra planning before actual coding begins will leave fewer problems to arise in the future.

We also tended to do a lot of our coding in the same room, which was also quite good. We were all able to resolve any issues that came up with other people's code on the spot. Because we were able to communicate so well there were few problems when it came time to integrate the different modules together. Finally, it is important to write code that others can understand. Comment liberally. During the integration phase I had little trouble fixing bugs in other group member's code when they were not around because they wrote very clear, readable code. This streamlines the integration process since everyone is able to fix bugs as they arise instead of being forced to wait for the person responsible for that piece of code to come and fix it.

8. Appendix

8.1. Code

8.1.1. Makefile

```
******************************
Makefile
******************************

JFLAGS = -g

%.class : %.java
        javac $(JFLAGS) $<

GRAMMAR_FILES = MapwadLexer.java MapwadParser.java MapwadVocabTokenTypes.java
GRAMMAR_CLASSES = MapwadVocabTokenTypes.class
WALKER_FILES = MapwadWalker.java MapwadWalkerTokenTypes.java MWAST.java
WALKER_CLASSES = MapwadWalkerTokenTypes.class MWAST.class

CLASSPATH := $(CLASSPATH):classes

all : Mapwad.class

class_dir : $(WALKER_CLASSES) $(GRAMMAR_CLASSES)
        cd classes && $(MAKE)

MapwadParser.java : grammar.g
        java antlr.Tool grammar.g
        touch $(GRAMMAR_FILES)

$(GRAMMAR_CLASSES) : MapwadParser.java
        javac $(JFLAGS) $(GRAMMAR_FILES)

MapwadWalker.java : walker.g $(GRAMMAR_CLASSES)
        java antlr.Tool walker.g
        touch $(WALKER_FILES)

$(WALKER_CLASSES) : MapwadWalker.java
        javac $(JFLAGS) $(WALKER_FILES)

Mapwad.class : Mapwad.java class_dir

clean:
        rm *.class
```

8.1.2. Lexer, Parser, Walker

```
******************************
grammar.g
******************************

/**
* grammar.g : The lexer and parser for MAPWAD written in ANTLR.
*
* @author MAPWAD Project
*
* @version $Id: grammar.g,v 1.46 2003/12/18 15:58:54 avrum Exp $
*/

class MapwadLexer extends Lexer;

options
{
  k = 2;
  charVocabulary = '\3'..'\377';
  testLiterals = false;
  exportVocab = MapwadVocab;
}

PLUS     : '+';
MINUS    : '-';
TIMES    : '*';
DIV      : '/';
MOD          : '%';

OR       : "||";
AND      : "&&";
NOT      : '!';
PLUSPLUS : "++";
MINMIN   : "--";

ASSIGN   : '=';
PLUSEQ   : "+=";
MINUSEQ  : "-=";
TIMESEQ  : "*=";
DIVEQ    : "/=";
MODEQ    : "%=";

EQ       : "==";
NE   : "!=";
GT       : '>';
LT       : '<';
GE       : ">=";
LE       : "<=";

PERIOD   : '.';
COMMA    : ',';
SEMI     : ';';
LPAREN   : '(';
RPAREN   : ')';
LCURL    : '{';
RCURL    : '}';
LBRACK   : '[';
RBRACK   : ']';

ARRAYTYPE: "[]";

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

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

// rule for C and C++ style comments (borrowed from Mx)
COMMENT  : ( "/*" (
                    options {greedy=false;} :
                    (NEWLINE)
                    | ~( '\n' | '\r' )
                 )* "*/"
            | "//" (~( '\n' | '\r' ))* (NEWLINE)
            )                      { $setType(Token.SKIP); }
         ;

ID options { testLiterals = true; }
         : ALPHA (ALPHA|DIGIT)*;
protected
ALPHA    : '_'|'a'..'z'|'A'..'Z';
protected
DIGIT    : '0'..'9';


/* Integers and floating point numbers similar to C */
NUM      : INT ((EXP)? | PERIOD INT (EXP)?);
protected
INT  : (DIGIT)+;
```

```
protected
EXP      : ('E'|'e') (PLUS | MINUS)? INT ;

//borrowed from Mx
STRING   : '"'!
              (  ~('"' | '\n')
               | ('"'!'"')
              )*
           '"'!
         ;

{
    import internal.MWException;
}

/**
 * MAPWAD Parser
 * @author Ben Smith (bhs16@columbia.edu)
 * @author Ron Weiss (ronw@cs.columbia.edu)
 */
class MapwadParser extends Parser;
options
{
    k = 3;
    buildAST = true;
    ASTLabelType = "MWAST";
    exportVocab = MapwadVocab;
    defaultErrorHandler=false;
}

tokens
{
    VAR_DEF;
    VAR_DEF_ASSIGN;
    ARRAY_CONST;
    ARRAY_ALLOC;
        FUNC_BODY;
    FUNC_CALL;
    FUNC_PARAM;
    PRE_UNARY;
}


/*
Basic structure of MAPWAD program:

global variable definitions

function definitions

room definitions

main map function: Map()
*/

program throws MWException,ANTLRException
{
    astFactory.setASTNodeClass("MWAST");
}
           : (definition)* EOF!
           ;

definition : ((type)? ID LPAREN) => func_def
           | var_def SEMI!
           ;

//NOTE: functions with no return values do not have to be declared void explicitly
func_def   : (type)? ID^ args_list func_body
           ;

//we're supporting default values for function arguments
```

```
args_list  : LPAREN! ( /*nothing*/ | var_def (COMMA! var_def)* ) RPAREN!
           ;

func_body  : LCURL! (statement)* RCURL!
                        { #func_body=#([FUNC_BODY,"FUNC_BODY"],func_body); }
           ;

var_def    : type ID^ (var_def_assign)?
            { #var_def=#([VAR_DEF,"VAR_DEF"],var_def);#var_def.setLineFromChild(); }
           ;

var_def_assign : ASSIGN! ( /*(type LBRACK) => array_alloc |*/ expression | array_const )
                 { #var_def_assign=#([VAR_DEF_ASSIGN,"VAR_DEF_ASSIGN"],var_def_assign); }
               ;

array_const : (LCURL! expression (COMMA! expression)* RCURL!)
             { #array_const=#([ARRAY_CONST,"ARRAY_CONST"],array_const); }
           ;

array_alloc : type (LBRACK! expression RBRACK!)+
             { #array_alloc=#([ARRAY_ALLOC,"ARRAY_ALLOC"],array_alloc); }
             ;

//parens after Map are optional since it can never take any arguments anyway
//map_def    : "Map"^ (LPAREN! RPAREN!)? func_body
//           ;

statement  : for_stmt
           | while_stmt
           | if_stmt
           | break_stmt
           | cont_stmt
           | return_stmt SEMI!
           | (type ID)=> var_def SEMI!
           | expression SEMI!
           ;


while_stmt : "while"^ LPAREN! expression RPAREN!
             LCURL! (statement)* RCURL!
           ;

//like C for statement
for_stmt   : "for"^ LPAREN! for_con RPAREN! LCURL!
             (statement)* RCURL!
           ;

for_con    : (expression | /*nothing*/) SEMI!
             (expression | /*nothing*/) SEMI!
             (expression | /*nothing*/);

break_stmt : "break" SEMI!;
cont_stmt  : "continue" SEMI!;
return_stmt: "return"^ (expression)?
           ;

if_stmt    : "if"^ LPAREN! expression RPAREN!
             func_body
             (else_stmt)?
           ;

else_stmt  : "else"^ (if_stmt | func_body)
           ;


//expressions follow precedence rules of C
expression : or ((PLUSEQ^|MINUSEQ^|TIMESEQ^|DIVEQ^|MODEQ^|ASSIGN^) expression)?;
or         : and (OR^ and)*;
and        : equal (AND^ equal)*;
equal      : cmp ((EQ^ | NE^) cmp)*;
cmp        : add ((GT^ | LT^ | GE^ | LE^) add)*;
```

```
add         : mult ((PLUS^ | MINUS^) mult)*;
mult        : sign ((TIMES^ | DIV^ | MOD^) sign)*;
sign        : ( (PLUS | MINUS | NOT) unary
              { #sign=#([PRE_UNARY,"PRE_UNARY"],sign); }
              )
             | unary;
unary       : bracket (PLUSPLUS^ | MINMIN^)?;
bracket     : (type LBRACK) => array_alloc | period (LBRACK^ expression RBRACK! (PERIOD^
atom)?  )*;
period      : atom (PERIOD^ atom)*;
atom        : NUM
            | STRING
            | "true"
            | "false"
            | ( ID LPAREN )=>func_call
            | ( type LPAREN )=>func_call
            | ID
            | map_start
            | map_height
            | ceiling_texture
            | floor_texture
            | LPAREN! expression RPAREN!
            ;

//supports optional arguments... eg f(), f(10) f(,,10), f(10,,9)
func_call  : (ID | obj_type)
              LPAREN! (expression)?  (COMMA | COMMA expression )*  RPAREN!
             { #func_call=#([FUNC_CALL,"FUNC_CALL"],func_call);
               #func_call.setLineFromChild(); }
            ;

/* legal types are:3 classes of types - primitive types, object types
 * primitives : int, float, boolean, string
 * objects    : Wall, Thing, Location, Room
 * user defined rooms
 * arrays of any of the above (eg. Wall[], array[]);
 */
//type       : ID (ARRAYTYPE)*
//           ;

type        : (prim_type | obj_type) (ARRAYTYPE)*
            ;

prim_type   : int_type | float_type | bool | string
            ;

obj_type    : room | wall | location | thing
            ;

//int and float need to have _type to avoid conflicts with Java versions
int_type    : "int";
float_type : "float";
bool        : "boolean";
string      : "string";
room        : "Room";
wall        : "Wall";
location    : "Location";
thing       : "Thing";
map_start   : "MapStart";    //Location
map_height : "MapHeight";   //float
//map_title  : "MapTitle";     //strin

ceiling_texture: "CeilingTexture"; //string
floor_texture  : "FloorTexture";   //string


******************************
walker.g
******************************

/*
```

```
* walker.g : The lexer and parser for MAPWAD written in ANTLR.
*
* @author Ben Smith (bhs16@cs.columbia.edu)
*
* @version $Id: walker.g,v 1.47 2003/12/17 21:14:12 bhs16 Exp $
*/

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

class MapwadWalker extends TreeParser;
options
{
    ASTLabelType = "MWAST";
    importVocab = MapwadVocab;
}

//This should be called first to get the function definitions and global variables
start throws MWException
{
       MWFunction f;
    MWDataType a;
}
    : (f=l:func                                    {
MWScope.s_global.addSymbol(f,#l.getLine()); }
       | a=l2:var_decl[MWScope.s_global]   { MWScope.s_global.addSymbol(a,#l2.getLine());
}
         )*
    ;

func returns [ MWFunction f ] throws MWException
{
    f=null;
    MWDataType ret=null,arg;
    LinkedList args=new LinkedList();
    AST fbody=null;
}
    : #( id:ID (ret=type[""])?
         (arg=var_decl[MWScope.s_global]     { args.add(arg);  }
         )*
          #(FUNC_BODY
            (bod:. {fbody=bod;})?            { f = new MWFunction(id.getText(),
                                                       id.getLine(),ret,args,fbody);
                                            }
           )
        )

       ;

var_decl [MWScope scope] returns [ MWDataType var ] throws MWException
{
    MWDataType a,b;
    LinkedList list;
       var=null;
}
       : #(VAR_DEF
        #(id:ID var=type[id.getText()]
           ( #(VAR_DEF_ASSIGN (a=expr[scope] ) )     { var.baseAssign(a,id.getLine());}

          )?
         )
       )
       ;

array_alloc [MWScope scope] returns [MWDataType array] throws MWException
{
    MWDataType a,b;
    LinkedList list=new LinkedList();
    array=null;
```

```
}
    : a=type[""] (b=l:expr[scope]                   { list.add(b); }
                 )*                                 {
array=MWArray.allocate(a,list,l.getLine()); }
    ;


type [String name] returns [ MWDataType r ]
{
    MWDataType var;
    int dims=0;
    r=null;
}
    : (
      "float"                           { var = new MWFloat(name); }
    | "int"                             { var = new MWInt(name); }
    | "string"                          { var = new MWString(name); }
    | "boolean"                         { var = new MWBoolean(name); }
    | "Room"                            { var = new MWReferenceType(name, new
MWRoom()); }
    | "Wall"                            { var = new MWReferenceType(name, new
MWWall()); }
    | "Location"                        { var = new MWReferenceType(name, new
MWLocation()); }
    | "Thing"                           { var = new MWReferenceType(name, new
MWThing()); }

       )
       ( l:ARRAYTYPE { dims++; } )*

    {
/*
        if( var instanceof MWWall && dims > 0)
        {
            var = new MWReferenceType(name, new MWWalls());
            dims--;
        }
*/
        if( dims > 0 )
        {
            r = MWArray.reference(name,var,dims);
        }
        else
            r=var;
    }
    ;


body [MWScope scope] throws MWException
{
    MWDataType a;
    scope.entering();
}
    : (
        if_stmt[scope]
      | for_stmt[scope]
      | while_stmt[scope]
      | a=pos:var_decl[scope]                   { scope.addSymbol(a,#pos.getLine()); }
      | a=expr[scope]
      )*
      ( l3:"continue"                         { MWContinue.throwContinue(l3.getLine());
}
      )?
      ( #(l1:"return"                         { a=null; }
          (a=expr[scope])? )
      {
          if(a==null)
                MWReturn.throwReturn(l1.getLine());
          else
                MWReturn.throwReturn(a,l1.getLine());
      }
      )?
      ( l2:"break"                             { MWBreak.throwBreak(l2.getLine()); }
      )?
{
    scope.leaving();
}
    ;

array_const [MWScope scope] returns [MWArray arr]  throws MWException
{
    MWDataType a;
    LinkedList list=new LinkedList();
    arr=null;
}
    : #(l:ARRAY_CONST
         (a=expr[scope]                        { list.add(a); }
         )*
       {
            arr = new MWArray((MWDataType)list.getFirst(),list.size());

            for(int i=0;i < list.size();i++)
                arr.elementAt(new
MWInt(i),l.getLine()).baseAssign((MWDataType)list.get(i),
                                l.getLine());
       }
       )
    ;

if_stmt [MWScope scope] throws MWException
{
    MWDataType result;
    MWBoolean test=new MWBoolean();
}
    : #("if" result=pos:expr[scope]              { test.baseAssign(result,pos.getLine());
}
        #(FUNC_BODY
          (ifbod:.
          {
             if(test.getValue())
                 body(ifbod,scope);
          }
          )?
         )
          (elsebod:.
          {
             if(!test.getValue())
                else_stmt(elsebod,scope);
          }
          )?
       )
    ;

else_stmt [MWScope scope] throws MWException
{
    MWDataType result;
    MWBoolean test=new MWBoolean(false);
}
    : #("else"
         ( if_stmt[scope]
          | #(FUNC_BODY bod3:. )                { body(bod3,scope); }
          )
      )
    ;

for_stmt [MWScope scope] throws MWException
{
    MWBoolean test=new MWBoolean();
    MWDataType temp;
}
    : #(l:"for" temp=expr[scope] eval:. incr:.
        (forbod:.
         {
            test.baseAssign(expr(eval,scope),l.getLine());

            while(test.getValue())
            {
                try
                {
                    body(forbod,scope);
                }
                catch(MWContinue c)
                { }
                catch(MWBreak b)
                { break; }

                expr(incr,scope);
                test.baseAssign(expr(eval,scope),l.getLine());
            }
         }
        )?
       )
    ;

while_stmt [MWScope scope] throws MWException
{
    MWBoolean test=new MWBoolean();
}
    : #(l:"while" eval:.
        (forbod:.
        {
            test.baseAssign(expr(eval,scope),l.getLine());
            while(test.getValue())
            {
                try
                {
                    body(forbod,scope);
                }
                catch(MWContinue c)
                { }
                catch(MWBreak b)
                { break; }

                test.baseAssign(expr(eval,scope),l.getLine());
            }
        }
        )?
       )
    ;

obj_type : "Room" | "Wall" | "Location" | "Thing"
         ;

builtin_vars : "MapStart" | "MapHeight" | "CeilingTexture" | "FloorTexture";

expr [MWScope scope] returns [ MWDataType r ] throws MWException
{
    MWDataType a,b;
    r=null;
}
    :
       ID                       { r=scope.getSymbol(#ID.getText(),#ID.getLine()); }
     | l:builtin_vars           { r=scope.getSymbol(l.getText(),l.getLine()); }
     | num:NUM
        {
            if(num.getText().indexOf('e') != -1
               || num.getText().indexOf('E') != -1
               || num.getText().indexOf('.') != -1)
            {
                r = new MWFloat(Float.parseFloat(num.getText()) );
            }
            else
            {
                r = new MWInt( Integer.parseInt(num.getText()));
            }
        }

    | "true"                          { r = new MWBoolean(true); }
    | "false"                   { r = new MWBoolean(false); }
    | str:STRING                { r = new MWString();

((MWString)r).setValue(str.getText());
                                }
    | #(ARRAY_ALLOC r=array_alloc[scope])
    | r=array_const[scope]
    //all commas are added as null
       | #(FUNC_CALL                { MWFunction f; }
            (ID                  { f =
(MWFunction)scope.getSymbol(#ID.getText(),#ID.getLine()); }
            | name:obj_type      { f =
(MWFunction)scope.getSymbol(name.getText(),name.getLine()); }
            )
                                 { LinkedList args=new LinkedList(); a=null; }
        ( (a=expr[scope]         { args.add(a); }
           | COMMA               { if(a == null)
                                   {
                                        args.add(a);
                                   }
                                   else
                                        a=null;    //just so it's not null
                                 }
          )
        )*
       )
      { r=f.execute(this,args,#FUNC_CALL.getLine()); }

    | #(LBRACK a=expr[scope] b=expr[scope] )   { r = a.elementAt(b,#LBRACK.getLine()); }
    | #(PERIOD a=expr[scope] ID)               { r =
a.getMember(#ID.getText(),#ID.getLine()); }
    | #(ASSIGN a=expr[scope] b=expr[scope])    { r = a.baseAssign(b,#ASSIGN.getLine()); }
    | #(LT a=expr[scope] b=expr[scope])        { r = a.lessThan(b,#LT.getLine()); }
    | #(GT a=expr[scope] b=expr[scope])        { r = a.greaterThan(b,#GT.getLine()); }
    | #(LE a=expr[scope] b=expr[scope])        { r = a.lessThanEq(b,#LE.getLine()); }
    | #(GE a=expr[scope] b=expr[scope])        { r = a.greaterThanEq(b,#GE.getLine()); }
    | #(EQ a=expr[scope] b=expr[scope])        { r = a.baseEquals(b,#EQ.getLine()); }
    | #(NE a=expr[scope] b=expr[scope])        { r = a.notEq(b,#NE.getLine()); }
    | #(PLUSEQ a=expr[scope] b=expr[scope])    { r = a.plusEq(b,#PLUSEQ.getLine()); }
    | #(MINUSEQ a=expr[scope] b=expr[scope])   { r = a.minusEq(b,#MINUSEQ.getLine()); }
    | #(TIMESEQ a=expr[scope] b=expr[scope])   { r = a.timesEq(b,#TIMESEQ.getLine()); }
    | #(DIVEQ a=expr[scope] b=expr[scope])     { r = a.divideEq(b,#DIVEQ.getLine()); }
    | #(MODEQ a=expr[scope] b=expr[scope])     { r = a.modEq(b,#MODEQ.getLine()); }
    | #(PLUSPLUS a=expr[scope])                { r = a.plusPlus(#PLUSPLUS.getLine()); }
    | #(MINMIN a=expr[scope])                  { r = a.minusMinus(#MINMIN.getLine()); }
    | #(PLUS a=expr[scope] b=expr[scope])      { r = a.add(b,#PLUS.getLine()); }
    | #(MINUS a=expr[scope] b=expr[scope])     { r = a.subtract(b,#MINUS.getLine()); }
    | #(TIMES a=expr[scope] b=expr[scope])     { r = a.multiply(b,#TIMES.getLine()); }
    | #(DIV a=expr[scope] b=expr[scope])       { r = a.divide(b,#DIV.getLine()); }
    | #(MOD a=expr[scope] b=expr[scope])       { r = a.mod(b,#MOD.getLine()); }
    | #(OR a=expr[scope] b=expr[scope])        { r = a.or(b,#OR.getLine()); }
    | #(AND a=expr[scope] b=expr[scope])       { r = a.and(b,#AND.getLine()); }
    | #(PRE_UNARY (
                    PLUS a=expr[scope]         { r = a.plusSign(#PLUS.getLine()); }
                  | MINUS a=expr[scope]        { r = a.minusSign(#MINUS.getLine()); }
                  | NOT a=expr[scope]          { r = a.not(#NOT.getLine()); }
                  )
       )
    ;


******************************
MWAST.java
******************************

import antlr.collections.AST;
import antlr.CommonAST;
import antlr.Token;

/**
 * Mapwad AST node
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWAST.java,v 1.3 2003/12/18 20:14:36 jdub Exp $
 */
public class MWAST extends CommonAST
{
    public MWAST()
    {}

    public MWAST(Token t)
    {
       super(t);
       m_line = t.getLine();
    }

    public void initialize(MWAST t)
    {
       super.initialize(t);
       m_line = t.getLine();
    }

    public void initialize(Token t)
    {
       super.initialize(t);
       m_line = t.getLine();
    }

    public int getLine()
    { return m_line; }

    public void setLineFromChild()
    {
       m_line = ((MWAST)getFirstChild()).getLine();
    }

    int m_line=-1;
}
```

8.1.3. Compiler Entry Code

```
******************************
mwc
******************************

#!/bin/sh
#Mapwad compiler:
#Author: Ron Weiss (ronw@cs.columbia.edu)
#
#1. calls Mapwad (java) to interpret mapwad files and generate .map file
#2. calls qbsp to compile .map file into .bsp file
#3. cleans up after qbsp
#
#
#How to play the map:
#1. copy the .bsp file into your quake/id1/maps directory
#2. run quake
#3. bring down console (~)
#4. type "map mapname" where mapname is the name of the .bsp file
#   without the .bsp extension (eg. to play maze.bsp, type
#   "map maze" in console)


if [ $1 = "-h" ]
then
```

```
    echo "Mapwad compiler: compiles Mapwad source into Quake 1 map"
    echo
    echo "USAGE:"
    echo "mwc [-h] [-m] [-g] sourcefile destfile"
    echo
    echo "-h    displays this help message"
    echo "-m    produces intermediate text .map file"
    echo "-g    only shows graphical view of map, doesn't compile to Quake map"
    echo
    echo "If no destfile is given, default destfile is used"
    echo "Eg: "
    echo "$ mwc test.map"
    echo "will produce test.bsp which can be loaded in Quake"
else
    keepmapfile=0
    noqbsp=0
    for x in $1 $2 $3 $4 $5 $6 $7 $8
    do
      if [ $x = "-m" ]
          then keepmapfile=1
      fi

      if [ $x = "-g" ]
          then noqbsp=1
      fi

    done

    if [ $keepmapfile -eq 1 ]
       then
       if [ $noqbsp -eq 1 ]
           then
           sourcefile=$3
           destfile=$4
       else
           sourcefile=$2
           destfile=$3
       fi
    else
       if [ $noqbsp -eq 1 ]
           then
           sourcefile=$2
           destfile=$3
       else
           sourcefile=$1
           destfile=$2
       fi
    fi



    TEMPFILE=`echo $sourcefile | sed 's/^[a-zA-Z0-9\/]*\/\([a-zA-Z0-9]*\).[a-zA-Z0-
9]*$/\1/'`
    mapfile=`echo "$TEMPFILE.map"`

    #generate .map file from .mapwad:
    ANTLRPATH=/home/ronw/plt/antlr-2.7.2/antlr.jar
    CLASSPATH=$ANTLRPATH:src:src/classes java Mapwad $sourcefile $mapfile


    #qbsp - convert text .map into usable .bsp

    #qbsp is dumb and looks for "quake" in the path, so we need to
    #change directories before calling it
    #fixed by editing qbsp source -Ron

    if [ $noqbsp = 0 ]
       then
       QBSPPATH=src/qutil
       $QBSPPATH/qbsp $mapfile
```

```
        if [ $destfile ]
            then
            if [ $destfile != "$TEMPFILE.bsp" ]
                then mv `echo "$TEMPFILE.bsp"` $destfile
            fi
        fi

    #get rid of the extra files qbsp generates that we're not interested in
        F=`echo "$TEMPFILE.h1"`
        if [ -e $F ]
            then rm -f $F
        fi

        F=`echo "$TEMPFILE.h2"`
        if [ -e $F ]
            then rm -f $F
        fi

        F=`echo "$TEMPFILE.pts"`
        if [ -e $F ]
            then rm -f $F
        fi

        F=`echo "$TEMPFILE.prt"`
        if [ -e $F ]
            then rm -f $F
        fi
    fi


    if [ $keepmapfile -eq 0 ]
        then rm $mapfile
    fi
fi


******************************
Mapwad.java
******************************

/*
 * Mapwad compiler
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: Mapwad.java,v 1.19 2003/12/18 15:31:51 ronw Exp $
 */

import java.io.*;
import java.util.*;
import antlr.*;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
import internal.*;

class Mapwad {
    private static LinkedList s_rooms = new LinkedList();
    private static LinkedList s_things = new LinkedList();

    public static void addRoom(MWRoom room)
    { s_rooms.add(room); }

    public static void addThing(MWThing thing)
    { s_things.add(thing); }

    static public void runMapStart(MapwadWalker walker, String filename)
        throws RecognitionException,MWException,IOException
    {
        MWFunction funcMap = (MWFunction)MWScope.s_global.getSymbol("Map");
    if(funcMap==null)
    {
        throw new MWException("The Map function is not defined");
    }

     funcMap.execute(walker,new LinkedList(),-1);

    Vector rooms = new Vector(s_rooms);
    Vector things = new Vector(s_things);
    MWReferenceType ref = (MWReferenceType)MWScope.s_global.getSymbol("MapStart");

    if(!ref.isInitialized())
        throw new MWException("MapStart<Location> must be defined");

    MWLocation mapStart = (MWLocation)ref.getObject();
    MWFloat mapHeight   = (MWFloat)MWScope.s_global.getSymbol("MapHeight");
    MWString ceilTex    = (MWString)MWScope.s_global.getSymbol("CeilingTexture");
    MWString floorTex   = (MWString)MWScope.s_global.getSymbol("FloorTexture");


    System.out.println("Rooms: "+rooms.size());
    System.out.println("Things: "+things.size());

    MWMap map = new MWMap(rooms,things,mapStart,mapHeight,ceilTex,floorTex);

    MWGui gui = new MWGui(map);

    map.createMapFile(filename);
}

public static void main(String[] args)
    throws FileNotFoundException,RecognitionException,TokenStreamException,
           InstantiationException,IllegalAccessException,
           IOException,ANTLRException
{
     try
    {
        MapwadLexer lexer;
        String outFile = "test.map";
        boolean showTree=false;

        if(args.length == 2)
        {
           lexer = new MapwadLexer(new DataInputStream(
                                   new FileInputStream(args[0])));
           outFile = args[1];

           if(outFile.equals("--tree"))
              showTree=true;
        }
        else if(args.length == 1)
           lexer = new MapwadLexer(new DataInputStream(
                                   new FileInputStream(args[0])));
        else
           lexer = new MapwadLexer(new DataInputStream(System.in));

         MapwadParser parser = new MapwadParser(lexer);
         parser.program();

        CommonAST ast = (CommonAST)parser.getAST();

        if(showTree)
        {
           ASTFrame frame = new ASTFrame("AST JTree Example",ast);
           frame.setVisible(true);
        }
        else
        {
           MWScope.addBuiltIns();

           MapwadWalker walker = new MapwadWalker();
```

```
                walker.start(ast);
                runMapStart(walker, outFile);
            }
        }
        catch(MWException e)
        {
            System.err.println(e.getMessage());
        }
        catch(ANTLRException e)
        {
             System.err.println(e);
        }
        catch(Exception e)
        {
            System.err.println("Internal compiler error");
        }
    }
}
```

8.1.4. Built-In Functions

```
******************************
MWFunction.java
******************************

import java.util.*;
import antlr.collections.AST;
import internal.*;

/**
 * Mapwad function class; stores information about a function
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWFunction.java,v 1.29 2003/12/18 19:33:12 avrum Exp $
 */
class MWFunction extends MWDataType
{
    MWDataType m_return;
    LinkedList m_args;
    AST m_body;
    int m_line; // line number function declared on

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

    public MWFunction(String name, int line, MWDataType ret, LinkedList args,AST body)
    {
       super(name);
       m_return = ret;
       m_args = args;
       m_body = body;
       m_line = line;
    }

    public AST getBody()
    {
       return m_body;
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args, int line)
       throws antlr.RecognitionException,MWException
    {
       MWScope scope = new MWScope();

       Iterator itDefArg = m_args.iterator();
       Iterator itArg = args.iterator();
```

```
        MWDataType arg,defArg;

        while(itDefArg.hasNext())
        {
            if(itArg.hasNext())
               arg = (MWDataType) itArg.next();
            else
               arg = null;

            defArg = (MWDataType) itDefArg.next();

            if( (arg == null && !defArg.isInitialized())
               || (arg != null && !defArg.isCompatibleWith(arg)) )
               errorInvalidArg(line);

            MWDataType localVar;

            try
            {
               localVar = (MWDataType)defArg.clone();
            }
            catch(CloneNotSupportedException e)
            {
               throw new MWException(e.getMessage());
            }

            if(arg == null)
               //add the default variable
               scope.addSymbol(localVar,m_line);
            else
            {
               localVar.baseAssign(arg,line);
               scope.addSymbol(localVar,line);
            }
        }

        MWDataType retval=null;
        MWReturn retex=null;

        try
        {
            if(m_body != null)
               walker.body(m_body,scope);
        }
        catch(MWReturn ret)
        {
            retval=ret.getValue();
            retex=ret;
        }

        if( m_return == null && retval != null )
            throw new MWException(retex.getLine(),getVarName()+"() - No declared return
type");

        else if( m_return != null && !(retval.isCompatibleWith(m_return)) )
            throw new MWException(retex.getLine(),getVarName()+"() - Invalid return type
<"
                                +retval.typeName()+">");

        return retval;
    }

    protected static void getNextArg(Iterator itArg, MWDataType var, int line)
        throws MWException
    {
        MWDataType temp;

        if(itArg.hasNext())
        {
            temp = (MWDataType)itArg.next();
            if(temp == null)
               var.setNull();
            else
               var.baseAssign(temp,line);
        }
        else
            var.setNull();
    }

    protected void errorNumArgs(int line) throws MWException
    {
       throw new MWException(line,getVarName()+"() - Incorrect number of arguments");
    }

    protected void errorInvalidArg(int line) throws MWException
    {
       throw new MWException(line,getVarName()+"() - Invalid argument");
    }
}


******************************
MWAdd.java
******************************

import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for Add() builtin
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWAdd.java,v 1.5 2003/12/18 19:33:10 avrum Exp $
 */
class MWAdd extends MWFunction
{
    public MWAdd()
    {
       super( "Add");
    }

    public MWDataType execute(MapwadWalker walker,
                             Collection args,int line)
               throws MWException
    {
       if( args.size() != 2 )
           errorNumArgs(line);

       Iterator itArg = args.iterator();
       MWRoom room=null;
       MWWall wall=null;

       try
       {
           room = (MWRoom)((MWReferenceType)itArg.next()).getObject();
           wall = (MWWall)((MWReferenceType)itArg.next()).getObject();
       }
       catch(ClassCastException e)
        {
           errorInvalidArg(line);
       }

       return new MWBoolean(room.getWalls().add(wall));
    }
}


******************************
MWAttach.java
******************************
```

```
import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for Attach() builtin
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWAttach.java,v 1.7 2003/12/18 19:33:11 avrum Exp $
 */
class MWAttach extends MWFunction
{
    public MWAttach()
    {
        super( "Attach");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
                throws MWException
    {
        if( args.size() != 2 && args.size() != 3 )
            errorNumArgs(line);

        Iterator itArg = args.iterator();
        MWWall a=null;
        MWWall b=null;
        MWString wallconnect=null;

        try
        {
            a = (MWWall)((MWReferenceType)itArg.next()).getObject();
            b = (MWWall)((MWReferenceType)itArg.next()).getObject();


            if(itArg.hasNext())
                wallconnect = (MWString)itArg.next();
            else
            {
                wallconnect = new MWString();
                wallconnect.setValue("center");  //default value
            }
        }
        catch(ClassCastException e)
         {
            errorInvalidArg(line);
        }

        return a.attach(b,wallconnect);
    }
}


******************************
MWClose.java
******************************

import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for Close() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWClose.java,v 1.3 2003/12/18 19:33:11 avrum Exp $
 */
class MWClose extends MWFunction
{
    public MWClose()
    {
        super( "Close");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
                throws MWException
    {
        if( args.size() != 1 )
            errorNumArgs(line);

        Iterator itArg = args.iterator();
        MWRoom a=null;

        try
        {
            a = (MWRoom)((MWReferenceType)itArg.next()).getObject();
        }
        catch(ClassCastException e)
         {
            errorInvalidArg(line);
        }

        return a.close();
    }
}


******************************
MWIsClosed.java
******************************

import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for IsClosed() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWIsClosed.java,v 1.3 2003/12/18 19:33:12 avrum Exp $
 */
class MWIsClosed extends MWFunction
{
    public MWIsClosed()
    {
        super( "IsClosed");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
                throws MWException
    {
        if( args.size() != 1 )
            errorNumArgs(line);

        Iterator itArg = args.iterator();
        MWRoom a=null;

        try
        {
            a = (MWRoom)((MWReferenceType)itArg.next()).getObject();
        }
        catch(ClassCastException e)
         {
            errorInvalidArg(line);
        }

        return a.isClosed();
    }
}
```

```
******************************
MWIsPath.java
******************************

import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for IsPath() builtin
 *
 *
 * @author Avrum
 * @version $Id: MWIsPath.java,v 1.6 2003/12/18 20:06:55 avrum Exp $
 */
class MWIsPath extends MWFunction
{
    public MWIsPath()
    {
       super( "IsPath");


    public MWDataType execute(MapwadWalker walker,
                           Collection args,int line)
              throws MWException
    {
       if( args.size() != 2 )
           errorNumArgs(line);

       Iterator itArg = args.iterator();
       MWDataType start=null;
       MWDataType end=null;

       MWRoom st=null;
       MWRoom en=null;

       try
       {
           start = ((MWReferenceType)itArg.next()).getObject();
           end =   ((MWReferenceType)itArg.next()).getObject();

           if(start instanceof MWWall)
              st = ((MWWall)start).getFromRoom();
           else if(start instanceof MWLocation)
              st = ((MWLocation)start).getNearWall().getFromRoom();
           else if(start instanceof MWRoom)
              st = (MWRoom)start;
           else
              errorInvalidArg(line);

           if(end instanceof MWWall)
              en = ((MWWall)end).getFromRoom();
           else if(end instanceof MWLocation)
              en = ((MWLocation)end).getNearWall().getFromRoom();
           else if(end instanceof MWRoom)
              en = (MWRoom)end;
           else
              errorInvalidArg(line);
       }
       catch(ClassCastException e)
        {
           errorInvalidArg(line);
       }

       return new MWBoolean(closed(st,en));
    }

    private boolean closed(MWRoom s, MWRoom e)
    {
       Vector rooms = new Vector();
```

```
        Vector closed = new Vector();

        closed.add(s);
        addChildren(s,rooms,closed);
        while(rooms.size()>0)
        {
            MWRoom test = (MWRoom)rooms.firstElement();
            rooms.remove(0);
            closed.add(test);
            if (test.equals(e))
                return true;
            addChildren(test,rooms,closed);
        }

        return false;
    }

    private void addChildren(MWRoom r, Vector rooms, Vector closed)
    {
        for(int i=0;i<r.getWalls().getNumWalls();i++)
        {
            MWWall temp = r.getWalls().getWall(i);
            if(temp.getIsEntry().getValue())
            {
                MWRoom from = temp.getConnectedTo().getFromRoom();
                if(!closed.contains(from))
                {
                    rooms.add(from);
                }
            }
        }

    }
}


******************************
MWMath.java
******************************

import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for MWCos() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWCos extends MWFunction
{
    public MWCos()
    {
        super("Cos");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
               throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;

        if(itArg.hasNext())
        {
            float x=0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {
                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWFloat((float)Math.cos(Math.toRadians(x)));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            throw new MWException("Incorrect number of parameters");

        return null;
    }
}


/*
 * MWFunction wrapper for MWSin() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWSin extends MWFunction
{
    public MWSin()
    {
        super("Sin");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
                throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {
                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWFloat((float)Math.sin(Math.toRadians(x)));
            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}


/*
 * MWFunction wrapper for MWTan() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWTan extends MWFunction
{
    public MWTan()
    {
        super("Tan");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
               throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
               x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {
               try
               {
                   x = ((MWInt)arg).getValue();
               }
               catch(ClassCastException e2)
               {
                   errorInvalidArg(line);
               }
            }

            i = new MWFloat((float)Math.tan(Math.toRadians(x)));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}


/*
 * MWFunction wrapper for MWAcos() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWAcos extends MWFunction
{
    public MWAcos()
    {
        super("Acos");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
               throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {
                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWFloat((float)Math.toDegrees(Math.acos(x)));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}


/*
 * MWFunction wrapper for MWAsin() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWAsin extends MWFunction
{
    public MWAsin()
    {
        super("Asin");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
               throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
               x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {
               try
               {
                   x = ((MWInt)arg).getValue();
               }
               catch(ClassCastException e2)
               {
                   errorInvalidArg(line);
               }
            }

            i = new MWFloat((float)Math.toDegrees(Math.asin(x)));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}


/*
 * MWFunction wrapper for MWAtan() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWAtan extends MWFunction
{
    public MWAtan()
    {
        super("Atan");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
               throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
               x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {
                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWFloat((float)Math.toDegrees(Math.atan(x)));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}


/*
 * MWFunction wrapper for MWSqrt() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWSqrt extends MWFunction
{
    public MWSqrt()
    {
        super("Sqrt");
    }

    public MWDataType execute(MapwadWalker walker,
                        Collection args,int line)
                throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {
                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWFloat((float)Math.sqrt(x));

            return i;
        }
        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}


/*
 * MWFunction wrapper for MWAbs() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWAbs extends MWFunction
{
    public MWAbs()
    {
        super("Abs");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
              throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {
                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWFloat((float)Math.abs(Math.toRadians(x)));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}


/*
 * MWFunction wrapper for MWExp() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWExp extends MWFunction
{
    public MWExp()
    {
        super("Exp");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
                throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {

                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWFloat((float)Math.exp(x));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}

/*
 * MWFunction wrapper for MWLog() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWLog extends MWFunction
{
    public MWLog()
    {
        super("Log");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
                throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;
        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {

                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWFloat((float)Math.log(x));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}


/*
 * MWFunction wrapper for MWPow() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWPow extends MWFunction
{
    public MWPow()
    {
        super("Pow");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
              throws MWException
    {
        Iterator itArg = args.iterator();
        MWFloat i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {

                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            if(itArg.hasNext())
            {
                float y=0;

                Object arg2 = itArg.next();
                try
                {
                    y = ((MWFloat)arg2).getValue();
                }
                catch(ClassCastException e)
                {
                    try
                    {
                        y = ((MWInt)arg2).getValue();
                    }
                    catch(ClassCastException e2)
                    {
                        errorInvalidArg(line);
                    }
                }

                i = new MWFloat((float)Math.pow(x, y));

                return i;

            }
        }

        //three or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}


/*
 * MWFunction wrapper for MWRound() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWRound extends MWFunction
{
    public MWRound()
    {
        super("Round");
    }

    public MWDataType execute(MapwadWalker walker,
                            Collection args,int line)
                throws MWException
    {
        Iterator itArg = args.iterator();
        MWInt i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {
                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWInt((int)Math.round(x));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}



/*
 * MWFunction wrapper for MWFloor() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWFloor extends MWFunction
{
    public MWFloor()
    {
        super("Floor");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
                throws MWException
    {
        Iterator itArg = args.iterator();
        MWInt i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {

                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWInt((int)Math.floor(x));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}


/*
 * MWFunction wrapper for MWCeiling() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWMath.java,v 1.2 2003/12/18 19:33:13 avrum Exp $
 */
class MWCeiling extends MWFunction
{
    public MWCeiling()
    {
        super("Ceiling");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
                throws MWException
    {
        Iterator itArg = args.iterator();
        MWInt i;

        if(itArg.hasNext())
        {
            float x = 0;

            Object arg = itArg.next();
            try
            {
                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {
                try
                {
                    x = ((MWInt)arg).getValue();
                }
                catch(ClassCastException e2)
                {
                    errorInvalidArg(line);
                }
            }

            i = new MWInt((int)Math.ceil(x));

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            errorNumArgs(line);

        return null;
    }
}
```

```
******************************
MWPrint.java
******************************

import java.util.Collection;
import internal.*;

/**
 * MWFunction wrapper for Print() builtin
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWPrint.java,v 1.7 2003/12/18 19:33:13 avrum Exp $
 */
class MWPrint extends MWFunction
{
    public MWPrint()
    {
       super( "Print");
    }

    public MWDataType execute(MapwadWalker walker,
                            Collection args,int line)
               throws MWException
    {
       if( args.size() != 1 )
           throw new MWException("Incorrect number of parameters");

       MWDataType data = (MWDataType) args.iterator().next();
       //if( ! (args.getFirst() instanceof MWString) )
       //    throw new MWException("Incorrect parameter");

       System.out.println( data.toString() );

       return data;
    }
}


******************************
MWRandFloat.java
******************************

import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for RandFloat() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWRandFloat.java,v 1.3 2003/12/18 19:33:13 avrum Exp $
 */
class MWRandFloat extends MWFunction
{
    public MWRandFloat()
    {
       super("RandFloat");
    }

    public MWDataType execute(MapwadWalker walker,
                            Collection args,int line)
               throws MWException
    {
       Iterator itArg = args.iterator();
       Random r = new Random();
       MWFloat i;

       if(itArg.hasNext())
```

```
        {
            float x;

            Object arg = itArg.next();
            try
            {

                x = ((MWFloat)arg).getValue();
            }
            catch(ClassCastException e)
            {

                x = ((MWInt)arg).getValue();
            }

            i = new MWFloat(x*r.nextFloat());

            return i;
        }

        //two or more arguments
        if(itArg.hasNext())
            throw new MWException("Incorrect number of parameters");

        //if we're here there were no arguments passed in
        i = new MWFloat(r.nextFloat());

        return i;
    }
}


******************************
MWRandInt.java
******************************

import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for RandInt() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWRandInt.java,v 1.4 2003/12/18 19:33:14 avrum Exp $
 */
class MWRandInt extends MWFunction
{
    private Random s_rand = new Random();

    public MWRandInt()
    {
        super("RandInt");
    }

    public MWDataType execute(MapwadWalker walker,
                              Collection args,int line)
            throws MWException
    {
        Iterator itArg = args.iterator();
        //Random r = new Random();
        MWInt i;

        if(itArg.hasNext())
        {
            int x = ((MWInt)itArg.next()).getValue();
            i = new MWInt(s_rand.nextInt(x));

            return i;
        }
        System.out.println("here");

        //two or more arguments
        if(itArg.hasNext())
            throw new MWException("Incorrect number of parameters");

        //if we're here there were no arguments passed in
        i = new MWInt(s_rand.nextInt());
        return i;
    }
}


******************************
MWRemove.java
******************************

import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for Remove() builtin
 *
 *
 * @author Ron Weiss (ronw@cs.columbia.edu)
 * @version $Id: MWRemove.java,v 1.3 2003/12/18 19:33:14 avrum Exp $
 */
class MWRemove extends MWFunction
{
    public MWRemove()
    {
        super( "Remove");
    }

    public MWDataType execute(MapwadWalker walker,
                             Collection args,int line)
               throws MWException
    {
        if( args.size() != 2 )
            errorNumArgs(line);

        Iterator itArg = args.iterator();
        MWRoom room=null;
        MWWall wall=null;

        try
        {
            room = (MWRoom)((MWReferenceType)itArg.next()).getObject();
            wall = (MWWall)((MWReferenceType)itArg.next()).getObject();
        }
        catch(ClassCastException e)
         {
            errorInvalidArg(line);
        }

        room.getWalls().remove(wall);

        //always return true
        return new MWBoolean(true);
    }
}
```

8.1.5. Reserved Word Wrappers

```
******************************
MWBreak.java
******************************

import internal.*;
```

```
/**
 * Mapwad class for break statements.
 * When the walker encounters a break statement it throws an MW.  This may not be ideal,
 * but I can't figure out a better way to get out of an ANTRL ()* loop.  Also, a break
 * statement isn't that different from an exception in terms of what it does, and
 * this approach has one advantage, which is that if a break isn't in an appropriate
 * block an exeception is "automatically" thrown.
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWBreak.java,v 1.3 2003/12/18 19:33:11 avrum Exp $
 */
class MWBreak extends MWException
{
    MWDataType m_value;

    public MWBreak(int line)
    {
       super(line,"'break' outside of loop");
    }

    // Here we're hiding the throw from Java. This is necessary because java
    // considers an unreachable statement an error.  ANTLR puts everything in switch
    // statements, so you can't throw an exception and not leave some way to reach
    // the break at the end of your case block
    public static void throwBreak(int line) throws MWBreak
    {
       throw new MWBreak(line);
    }
}


******************************
MWContinue.java
******************************

import internal.*;

/**
 * Mapwad class for continue statements
 * When the walker encounters a continue statement it throws an MWContinue.
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWContinue.java,v 1.2 2003/12/18 19:33:11 avrum Exp $
 */
class MWContinue extends MWException
{
    MWDataType m_value;

    public MWContinue(int line)
    {
       super(line,"'continue' outside of loop");
    }

    //Here we're hiding the throw from Java. This is necessary because java
    //considers an unreachable statement an error.  ANTLR puts everything in switch
    //statements, so you can't throw an exception and not leave some way to reach
    //the break at the end of your case block
    public static void throwContinue(int line) throws MWContinue
    {
       throw new MWContinue(line);
    }
}


******************************
MWReturn.java
******************************

import internal.*;

/**
```

```
 * Mapwad class for return values
 * When the walker encounters a return statement it throws an MWReturn.  This may not be
ideal,
 * but I can't figure out a better way to get out of an ANTRL ()* loop. Also, a break
 * statement isn't that different from an exception in terms of what it does, and
 * this approach has one advantage, which is that if a break isn't in an appropriate
 * block an exeception is "automatically" thrown.
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWReturn.java,v 1.4 2003/12/18 19:33:14 avrum Exp $
 */
class MWReturn extends MWException
{
    MWDataType m_value;

    public MWReturn(MWDataType value, int line)
    {
       super(line,"'return' outside of function");
       m_value=value;
    }

    public MWDataType getValue()
    { return m_value; }

    //Here we're hiding the throw from Java. This is necessary because java
    //considers an unreachable statement an error.  ANTLR puts everything in switch
    //statements, so you can't throw an exception and not leave some way to reach
    //the break at the end of your case block
    public static void throwReturn(MWDataType a,int line) throws MWReturn
    {
       throw new MWReturn(a,line);
    }

    public static void throwReturn(int line) throws MWReturn
    {
       throw new MWReturn(null,line);
    }
}
```

8.1.6. Object Constructor Wrappers

```
******************************
MWLocationConstruct.java
******************************

import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for Location constructor
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWLocationConstruct.java,v 1.5 2003/12/18 19:33:13 avrum Exp $
 */
class MWLocationConstruct extends MWFunction
{
    public MWLocationConstruct()
    {
       super("Location");
    }

    public MWDataType execute(MapwadWalker walker,
                            Collection args,int line)
               throws MWException
    {
       if( args.size() < 1 )
           errorNumArgs(line);

       Iterator itArg = args.iterator();
```

```
        MWReferenceType nearWall = new MWReferenceType(new MWWall());
        //MWWall nearWall = new MWWall();
        MWString wallPosition = new MWString();
        MWFloat wallDistance = new MWFloat();   //Could be float or int

        MWLocation location = new MWLocation();

        getNextArg(itArg,nearWall,line);
        getNextArg(itArg,wallPosition,line);
        getNextArg(itArg,wallDistance,line);

        if(!nearWall.isNull())
            location.setNearWall((MWWall)nearWall.getObject());
        else
            errorInvalidArg(line);

        if(!wallPosition.isNull())
            location.setWallPosition(wallPosition);

        if(!wallDistance.isNull())
            location.setWallDistance(wallDistance);

        if(itArg.hasNext())
            errorNumArgs(line);

        return new MWReferenceType(location);
    }
}


******************************
MWRoomConstruct.java
******************************

import internal.*;
import java.util.*;

/**
 * MWFunction wrapper for Room constructor
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWRoomConstruct.java,v 1.9 2003/12/18 20:00:27 avrum Exp $
 */
class MWRoomConstruct extends MWFunction
{
    public MWRoomConstruct()
    {
        super("Room");
    }

    public MWDataType execute(MapwadWalker walker,
                            Collection args,int line)
               throws MWException
    {
        if( args.size() != 0 )
            errorNumArgs(line);

        MWRoom room = new MWRoom();

        Mapwad.addRoom(room);

        return new MWReferenceType(room);
    }
}


******************************
MWThingConstruct.java
******************************
```

```
import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for Thing constructor
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWThingConstruct.java,v 1.8 2003/12/18 19:33:15 avrum Exp $
 */
class MWThingConstruct extends MWFunction
{
    public MWThingConstruct()
    {
        super("Thing");
    }

    public MWDataType execute(MapwadWalker walker,
                             Collection args,int line)
               throws MWException
    {
        if( args.size() > 2 )
            errorNumArgs(line);

        Iterator itArg = args.iterator();
        MWString thingType=null;
        MWLocation location=null;
        MWThing thing = new MWThing();

        try
        {
            if(itArg.hasNext())
            {
               thingType = (MWString)itArg.next();

               if(thingType != null)
                  thing.setType(thingType);
            }

            if(itArg.hasNext())
            {
                location = (MWLocation)((MWReferenceType)itArg.next()).getObject();
                //location = (MWLocation)itArg.next();

                if(location != null)
                    thing.setPosition(location);
            }
        }
        catch(ClassCastException e)
         {
            errorInvalidArg(line);
        }

        Mapwad.addThing(thing);

        return new MWReferenceType(thing);
    }
}


******************************
MWWallConstruct.java
******************************

import java.util.*;
import internal.*;

/**
 * MWFunction wrapper for Wall constructor
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWWallConstruct.java,v 1.8 2003/12/18 19:33:15 avrum Exp $
 */

class MWWallConstruct extends MWFunction
{
    public MWWallConstruct()
    {
       super("Wall");
    }

    public MWDataType execute(MapwadWalker walker,
                             Collection args,int line)
               throws MWException
    {
       if( args.size() == 0 )
           return new MWReferenceType(new MWWall());

       MWDataType temp;

       MWFloat length=new MWFloat();
       MWFloat angletonext=new MWFloat();
       MWString name = new MWString();
       MWString texture = new MWString();

       Iterator itArg = args.iterator();

       try
        {
           getNextArg(itArg,length,line);
           getNextArg(itArg,angletonext,line);
           getNextArg(itArg,name,line);
           getNextArg(itArg,texture,line);
       }
       catch(MWException e)
       {
           errorInvalidArg(line);
       }

       MWWall wall = new MWWall();

       if( !length.isNull() )
           wall.setLength(length);

       if( !angletonext.isNull() )
           wall.setAngleToNext(angletonext);

       if( !name.isNull() )
           wall.setName(name);

       if( !texture.isNull() )
           wall.setTexture(texture);

       if(itArg.hasNext())
           errorNumArgs(line);

       return new MWReferenceType(wall);
    }

    protected static void getNextArg(Iterator itArg, MWDataType var, int line)
       throws MWException
    {
       MWDataType temp;

       if(itArg.hasNext())
       {
           temp = (MWDataType)itArg.next();

           if(temp == null)
              var.setNull();
           else
```

```
                var.assign((MWDataType)temp,line);
        }
        else
            var.setNull();
    }
}
```

8.1.7. Scope Handler

```
******************************
MWScope.java
******************************

import java.util.*;
import internal.*;

/**
 * Mapwad scope class
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWScope.java,v 1.21 2003/12/18 19:33:15 avrum Exp $
 */
class MWScope
{
    HashMap m_current=null;
    LinkedList m_symbols;

    public static MWScope s_global = new MWScope();

    public MWScope()
    {
       m_symbols = new LinkedList();
       m_current = new HashMap();
       m_symbols.add(m_current);
    }

    public MWScope(MWScope parent)
    {
       m_symbols = new LinkedList(parent.m_symbols);
       m_current = new HashMap();
       m_symbols.add(m_current);
    }

    public void addSymbol(MWDataType var, int line) throws MWException
    {
       if(getSymbol(var.getVarName()) != null)
           throw new MWException(line,"Redefinition of " + var.getVarName());

       m_current.put(var.getVarName(),var);
    }

    public void addSymbol(String name, MWDataType var,int line) throws MWException
    {
       if(getSymbol(name) != null)
           throw new MWException(line,"Redefinition of " + name);

       m_current.put(name,var);
    }

    protected MWDataType getSymbol(String str)
    {
       MWDataType var=null;
       HashMap sym;

       ListIterator itSym = m_symbols.listIterator();

       while(itSym.hasNext())
       {
           sym = (HashMap)itSym.next();
```

```
        var = (MWDataType)sym.get(str);

        if(var != null)
           return var;
    }

    if(this != s_global)
        return s_global.getSymbol(str);
    else
        return null;
}

public MWDataType getSymbol(String str, int line) throws MWException
{
    MWDataType var = getSymbol(str);

    if(var == null)
        throw new MWException(line,"Undefined symbol : "+str);

    return var;
}

public void entering()
{
    m_current = new HashMap();
    m_symbols.add(m_current);
}

public void leaving()
{
    m_symbols.removeLast();
    m_current=(HashMap)m_symbols.getLast();
}

public static void addBuiltIns()
    throws MWException
{
    //Constructors
    s_global.addSymbol(new MWRoomConstruct(),-1);
    s_global.addSymbol(new MWWallConstruct(),-1);
    s_global.addSymbol(new MWThingConstruct(),-1);
    s_global.addSymbol(new MWLocationConstruct(),-1);

    //Built-in functions
    s_global.addSymbol(new MWAdd(),-1);
    s_global.addSymbol(new MWAttach(),-1);
    s_global.addSymbol(new MWPrint(),-1);
    s_global.addSymbol(new MWRandInt(),-1);
    s_global.addSymbol(new MWRandFloat(),-1);
    s_global.addSymbol(new MWClose(),-1);
    s_global.addSymbol(new MWIsClosed(),-1);
    s_global.addSymbol(new MWRemove(),-1);
    s_global.addSymbol(new MWIsPath(),-1);

    //Math functions
    s_global.addSymbol(new MWCos(),-1);
    s_global.addSymbol(new MWSin(),-1);
    s_global.addSymbol(new MWTan(),-1);
    s_global.addSymbol(new MWAcos(),-1);
    s_global.addSymbol(new MWAsin(),-1);
    s_global.addSymbol(new MWAtan(),-1);
    s_global.addSymbol(new MWAbs(),-1);
    s_global.addSymbol(new MWSqrt(),-1);
    s_global.addSymbol(new MWExp(),-1);
    s_global.addSymbol(new MWLog(),-1);
    s_global.addSymbol(new MWPow(),-1);
    s_global.addSymbol(new MWRound(),-1);
    s_global.addSymbol(new MWFloor(),-1);
    s_global.addSymbol(new MWCeiling(),-1);
```

```
        //Math constants
        s_global.addSymbol(new MWFloat("PI", (float)Math.PI),-1);
        s_global.addSymbol(new MWFloat("E", (float)Math.E),-1);


        //Built-in variables
        MWDataType nullType = new MWDataType("null");
        nullType.setNull();
        s_global.addSymbol(nullType,-1);
        s_global.addSymbol(new MWReferenceType("MapStart",new MWLocation()),-1);
        s_global.addSymbol(new MWFloat("MapHeight"),-1);

        s_global.addSymbol(new MWString("CeilingTexture"),-1);
        s_global.addSymbol(new MWString("FloorTexture"),-1);
    }
}
```

8.1.8. Mapwad Data Type Handling

```
******************************
MWDataType.java
******************************

package internal;

/**
 * Base class for all Mapwad data types
 *
 * (adapted from MX's MxDataType)
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWDataType.java,v 1.20 2003/12/18 19:32:35 avrum Exp $
 */

public class MWDataType implements Cloneable
{
    String m_varName="";
    boolean m_initialized=false;
    boolean m_readOnly=false;
    boolean m_null=false;

    public MWDataType()
    {}

    public MWDataType(String varName)
    {
       m_varName=varName;
    }

     /**
     * Extending classes should override to return a human readeable
     * string name.  Used mainly for error messages.
     *
     * @return String containing name of data type
     */
    public String typeName()
    { return "unknown"; }

    public String getVarName()
    { return m_varName; }

    void setReadOnly()
    { m_readOnly=true; }

    public Object clone() throws CloneNotSupportedException
    {
       return super.clone();
    }

    public boolean isCompatibleWith(MWDataType type)
```

```
{
   return false;
}

public void setInitialized()
{ m_initialized=true; }

public boolean isInitialized()
{ return m_initialized; }

public boolean isNull()
{ return m_null; }

public void setNull()
{ m_null = true; }

public MWDataType assign(MWDataType data, int line) throws MWException
{ return error(data,"=", line); }

public MWDataType baseAssign(MWDataType data, int line) throws MWException
{
   m_initialized=true;

   if(m_readOnly)
       throw new MWException(line,m_varName+" is read-only");

   if( !data.isNull() )
   {
       m_null=false;

       if( !isCompatibleWith(data) )
          return error(data,"=", line);
       else
          return assign(data,line);
   }
   else
   {
       m_null=true;
       return this;
   }
}

public MWDataType or(MWDataType data, int line) throws MWException
{ return error(data,"||", line); }

public MWDataType and(MWDataType data, int line) throws MWException
{ return error(data,"&&", line); }

public MWDataType not(int line) throws MWException
{ return error("!", line); }

public MWDataType baseEquals(MWDataType data, int line) throws MWException
{
   if(data.isNull())
       if(isNull())
          return new MWBoolean(true);
       else
          return new MWBoolean(false);

   if( !isCompatibleWith(data) )
       error(data,"==", line);

   return equals(data,line);
}

public MWDataType equals(MWDataType data, int line) throws MWException
{ return error(data,"==", line); }

public MWDataType notEq(MWDataType data, int line) throws MWException
{ return error(data,"!=", line); }
```

```
public MWDataType plusEq(MWDataType data, int line) throws MWException
{ return error(data,"+=", line); }

public MWDataType minusEq(MWDataType data, int line) throws MWException
{ return error(data,"-=", line); }

public MWDataType timesEq(MWDataType data, int line) throws MWException
{ return error(data,"*=", line); }

public MWDataType divideEq(MWDataType data, int line) throws MWException
{ return error(data,"/=", line); }

public MWDataType modEq(MWDataType data, int line) throws MWException
{ return error(data,"%=", line); }

public MWDataType lessThan(MWDataType data, int line) throws MWException
{ return error(data,"<", line); }

public MWDataType greaterThan(MWDataType data, int line) throws MWException
{ return error(data,">", line); }

public MWDataType lessThanEq(MWDataType data, int line) throws MWException
{ return error(data,"<=", line); }

public MWDataType greaterThanEq(MWDataType data, int line) throws MWException
{ return error(data,">=", line); }

public MWDataType plusPlus(int line) throws MWException
{ return error("++", line); }

public MWDataType minusMinus(int line) throws MWException
{ return error("++", line); }

public MWDataType plusSign(int line) throws MWException
{ return error("+", line); }

public MWDataType minusSign(int line) throws MWException
{ return error("-", line); }

public MWDataType add(MWDataType data, int line) throws MWException
{ return error(data,"+", line); }

public MWDataType subtract(MWDataType data, int line) throws MWException
{ return error(data,"-", line); }

public MWDataType multiply(MWDataType data, int line) throws MWException
{ return error(data,"*", line); }

public MWDataType divide(MWDataType data, int line) throws MWException
{ return error(data,"/", line); }

public MWDataType mod(MWDataType data,int line) throws MWException
{ return error(data,"%", line); }

public MWDataType getMember(String member, int line) throws MWException
{
   MWDataType var=getMember(member);

   if(var == null)
   {
       throw new MWException(line,m_varName+"<"+typeName()+">"
                            +"does not have a member variable "
                            +member);
   }

   return var;
}

public MWDataType getMember(String member)
{
   return null;
```

```
    }

    public MWDataType elementAt(MWDataType data, int line) throws MWException
    { return error(data,"[]",line); }

    public MWDataType error(MWDataType data, String op, int line) throws MWException
    {
        throw new MWException(line,
                         "Illegal operation: '" + op + "' on "
                          + m_varName
                          + " <" + typeName() + "> and "
                          +  data.m_varName
                          + " <" + data.typeName() + ">");
    }

    public MWDataType error(String op, int line) throws MWException
    {
        throw new MWException(line,
                         "Illegal operation: '" + op + "' on "
                          + m_varName
                          + " <" + typeName() + ">");
    }
}


******************************
MWReferenceType.java
******************************

package internal;

/**
 * Mapwad reference type
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWReferenceType.java,v 1.6 2003/12/18 19:32:37 avrum Exp $
 */

public class MWReferenceType extends MWDataType
{
    MWDataType m_object;

    public MWDataType getObject()
    {
        if (isNull())
            return null;
        else
            return m_object;
    }

    void setObject(MWDataType object)
    { m_object = object; }

    public String typeName()
    { return m_object.typeName(); }

    public boolean isCompatibleWith(MWDataType type)
    {
        if(type instanceof MWReferenceType &&
           m_object.isCompatibleWith(((MWReferenceType)type).m_object) )
            return true;

        return false;
    }

   /**
     * Constructor for a Location reference.
     *  @param name The Mapwad variable name
     */
    public MWReferenceType(String name, MWDataType type)
    {
       super(name);
       m_object=type;
    }

    public MWReferenceType(MWDataType object)
    {
       m_object=object;
       isInitialized();
    }

   /**
     * Assignment operator.  Called by MapwadWalker
     *  @param data The data type to be assigned to
     *  @param line The line number where the assignment occured
     */
    public MWDataType assign(MWDataType data, int line) throws MWException
    {
       m_object = ((MWReferenceType)data).m_object;

       return this;
    }

    public MWDataType equals(MWDataType data, int line) throws MWException
    {
       if( !isCompatibleWith(data) )
           return error(data,"==",line);

       return new MWBoolean( ((MWReferenceType)data).getObject() == m_object);
    }

    public MWDataType notEq(MWDataType data, int line) throws MWException
    {
       if( !isCompatibleWith(data) )
           return error(data,"!=",line);

       return new MWBoolean( ((MWReferenceType)data).getObject() != m_object);
    }

    public MWDataType getMember(String memberType)
    {
       return m_object.getMember(memberType);
    }
}
```

8.1.9. Advanced Types

```
******************************
MWArray.java
******************************

package internal;

import java.util.*;

/**
 * Mapwad multi-dimensional array. Stores array of Mapwad data types
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWArray.java,v 1.18 2003/12/18 20:00:37 avrum Exp $
 */

public class MWArray extends MWDataType
{
    //The type of this array
    MWDataType m_type;
    MWDataType[] m_data=null;

    public String typeName()
```

```
{
   return m_type.typeName()+"[]";
}

MWArray()
{
}

public MWArray(String name, MWDataType type)
{
   super(name);
   m_type=type;
}

public MWArray(String name, MWDataType type, int size) throws MWException
{
   super(name);

   m_type = type;

   setSize(size);
}

public MWArray(MWDataType type, int size) throws MWException
{
   m_type = type;

   setSize(size);

   setInitialized();
}

public MWArray(MWDataType type)
{
   m_type = type;
}

public boolean isCompatibleWith(MWDataType type)
{
   if( type instanceof MWArray
       && ((MWArray)type).m_type.isCompatibleWith(m_type))
       return true;

   return false;
}

public MWDataType getType()
{ return m_type; }

//This will wipe out anything in the array.  Only intended to be called once.
void setSize(int size) throws MWException
{
   setInitialized();

   m_data = new MWDataType[size];

   try
   {
       for(int i=0;i < m_data.length;i++)
       {
          if(m_type instanceof MWReferenceType)
          {
              MWReferenceType data =
                 new MWReferenceType(null,((MWReferenceType)m_type).getObject());

              m_data[i] = data;
          }
          else
              m_data[i]=(MWDataType)m_type.getClass().newInstance();
```

```
                if( m_type instanceof MWArray )
                    ((MWArray)m_data[i]).m_type = m_type;
            }
        }
        catch(InstantiationException e)
        {
            throw new MWException("INTERNAL ERROR : Could not instantiate array for type
<"
                                  +m_type.typeName()+">");
        }
        catch(IllegalAccessException e)
        {
            throw new MWException("INTERNAL ERROR : Could not instantiate array for type
<"
                                  +m_type.typeName()+">");
        }
    }

    public MWDataType elementAt(MWDataType index, int line) throws MWException
    {
        if( !(index instanceof MWInt) )
            throw new MWException(line,
                      "Attempt to reference array element with non-integer value");

        if(!isInitialized())
            throw new MWException(line,"Array has not been initialized");

        if(((MWInt)index).m_value > m_data.length-1 || ((MWInt)index).m_value < 0)
            throw new MWException(line,"Array index out of bounds");

        return m_data[((MWInt)index).m_value];
    }

    public MWDataType assign(MWDataType data,int line) throws MWException
    {
        if( data instanceof MWArray && m_type.isCompatibleWith( ((MWArray)data).m_type ) )
        {
            m_data = ((MWArray)data).m_data;

            return this;
        }

        return error(data,"=",line);
    }

    private static MWArray allocDim(MWDataType type, MWDataType[] dims, int index, int
line) throws MWException
    {
        MWArray array;
        int i;

        MWDataType size = dims[index];

        if( !(size instanceof MWInt) )
            errorNonIntDimension(line);

        if(index < dims.length-1)
        {
            array=reference("",type,dims.length-index);
            array.setSize(((MWInt)size).getValue());

            for(i=0;i < array.m_data.length;i++)
            {
                array.m_data[i]=allocDim(type,dims,index+1,line);
            }
        }
        else
        {
            array = new MWArray(type,((MWInt)size).getValue());
        }
        return array;
    }

    //returns an array allocated with the dimensions specified in dims
    public static MWArray allocate(MWDataType type, Collection dims, int line) throws
MWException
    {
        MWDataType[] dimsArray = (MWDataType[]) dims.toArray(new MWDataType[1]);

        return allocDim(type,dimsArray,0,line);
    }

    /**
     * Creates an array reference
     *  @param name The Mapwad variable name
     *  @param type The object type of the array reference
     *  @param dims The number of dimensions
     */
     public static MWArray reference(String name, MWDataType type, int dims)
    {
        MWArray array = new MWArray(name,type);

        for(int i=1; i < dims;i++)
         {
            array = new MWArray(name,array);
        }

        return array;
    }

    public Object clone() throws CloneNotSupportedException
    {
        MWArray theClone = (MWArray)super.clone();

        if(isNull())
            return theClone;

        try
        {
            if(m_data != null)
            {
                theClone.setSize(m_data.length);

                for(int i=0; i < m_data.length;i++)
                    theClone.m_data[i] = (MWDataType)m_data[i].clone();
            }
        }
        catch(MWException e)
        {
            throw new CloneNotSupportedException(e.getMessage());
        }

        return theClone;
    }

    public static final void errorNonIntDimension(int line) throws MWException
    {
        throw new MWException(line,"Array dimensions must be <int>");
    }

    public static final void errorWrongIndexType(MWDataType index,int line) throws
MWException
    {
        throw new MWException(line,"Cannot index array with <"+index.typeName()+">");
    }

    /**
     * Returns the member variables associated with this object.
     *
     * @param memberType The type of member variable to be returned.
     * @return The member variable desired.
     */
    public MWDataType getMember(String memberType)
    {
       if(memberType.equals("Size"))
       {
           return new MWInt(m_data.length);
       }

       return null;
    }

}


******************************
MWLocation.java
******************************

package internal;

import java.awt.geom.*;

/**
 * The basic Location object in MAPWAD.
 *
 * A Location is a three-tuple consisting of a Wall object, a string
 * representation of a position relative to the Wall, and a distance
 * perpendicularly outward into the Room from the Wall.
 *
 * @author Avrum Tilman
 * @version CVS $Id: MWLocation.java,v 1.18 2003/12/18 19:32:37 avrum Exp $
 */

public class MWLocation extends MWDataType
{
    public static final MWWall D_NEAR_WALL=null;
    public static final String D_WALL_POSITION="center";
    public static final float D_WALL_DISTANCE=5;
    public static final float D_Z_COORD=25;

    //private MWWall m_NearWall;
    private MWReferenceType m_NearWall;
    private MWFloat m_WallDistance;
    private MWString m_WallPosition;
    private Point2D.Float m_coord = null;

   /**
     * Constructor for a Location reference.
     *
     * When a variable is constructed with a name, it has not been
     * instantiated yet, so we don't need to do any intialization here.
     *
     * @param name The Mapwad variable name.
     */
    public MWLocation(String name)
    {
       super(name);
    }

    /**
     * The default constructor for a Location Object.
     */
    public MWLocation()
    {
       m_NearWall = new MWReferenceType(new MWWall());
       m_WallDistance=new MWFloat(D_WALL_DISTANCE);
       m_WallPosition=new MWString();
       m_WallPosition.setValue(D_WALL_POSITION);
    }

    /**
     * The constructor for a Location Object.
     *
     * @param near The wall this location is near.
     * @param pos Whether the location is in the center of the wall, to
     * the left of the wall, or to the right of the wall.
     * @param dist How far away from the wall is the location.
     */
    public MWLocation(MWWall near, MWString pos, MWFloat dist)
    {
       m_NearWall.setObject(near);
       m_WallDistance=dist;
       m_WallPosition=pos;
    }


    /**
     * Get function for the NearWall member variable.
     *
     * @return MWWall The Wall that this Location's position is based on.
     */
    public MWWall getNearWall()
    {
       return (MWWall)m_NearWall.getObject();
    }

    /**
     * Get function for the WallDistance member variable.
     *
     * @return MWFloat The distance away from the wall of the Location.
     */
    public MWFloat getWallDistance()
    {
       return m_WallDistance;
    }

    /**
     * Get function for the WallPosition member variable.
     *
     * @return MWString The position on the Wall of the Location.
     */
    public MWString getWallPosition()
    {
       return m_WallPosition;
    }

    /**
     * Set function for the NearWall member variable.
     *
     * @param near The Wall near the location.
     */
    public void setNearWall(MWWall near)
    {
       m_NearWall.setObject(near);
    }

    /**
     * Set function for the WallDistance member variable.
     *
     * @param dist The distance from the Wall of the Location.
     */
    public void setWallDistance(MWFloat dist)
    {
       m_WallDistance=dist;
    }

    /**
     * Set function for the WallPosition member variable.
     *
     * @param pos The position on the Wall of the Location.
     */
    public void setWallPosition(MWString pos)
```

```
{
   m_WallPosition=pos;
}

/**
 * Set function for the coord member variable.
 *
 * @param x x coordinate of location
 * @param y y coordinate of location
 */
void setCoord(float x, float y)
{
   m_coord = new Point2D.Float(x, y);
}

/**
 * Set function for the coord member variable.
 *
 * @param p point containing coordinates of location
 */
void setCoord(Point2D.Float p)
{
   m_coord = p;
}

/**
 * Get function for the coord member variable.
 *
 * @return point containing coordinates of location
 */
Point2D.Float getCoord()
{
   return(m_coord);
}

public String toString()
{
   if(m_coord != null)
       return("\""+Math.round(m_coord.getX())+" "
             +Math.round(m_coord.getY())+" "
             +Math.round(D_Z_COORD)+"\"");
   else
       return("");
}

/*
 * The following functions are used during the preprocessing phase.
 */

/**
 * Returns the type of data type.
 *
 * @return The type of data type.
 */
public String typeName()
{
   return "Location";
}

/**
 * Whether or not the specified data type is compatible with
 * an MWLocation.
 *
 * @param type The data type in question.
 * @return Whether or not the specified type is compatible.
 */
public boolean isCompatibleWith(MWDataType type)
{
   if(type instanceof MWLocation)
   {
       return true;
```

```
        }

        return false;
    }

   /**
     * Assignment operator.
     *
     * Called by MapwadWalker.
     *
     * @param data The data type to be assigned to.
     * @param line The line number where the assignment occured.
     */
    public MWDataType assign(MWDataType data, int line) throws MWException
    {
        m_NearWall = ((MWLocation)data).m_NearWall;
        m_WallDistance = ((MWLocation)data).m_WallDistance;
        m_WallPosition = ((MWLocation)data).m_WallPosition;

        return this;
    }

    /**
     * Returns the member variables associated with this object.
     *
     * @param memberType The type of member variable to be returned.
     * @return The member variable desired.
     */
    public MWDataType getMember(MWString memberType)
    {
        if(memberType.getValue().equals("NearWall"))
        {
            return m_NearWall;
        }
        if(memberType.getValue().equals("WallDistance"))
        {
            return m_WallDistance;
        }
        if(memberType.getValue().equals("WallPosition"))
        {
            return m_WallPosition;
        }
        return null;
    }
}


******************************
MWRoom.java
******************************

package internal;

import java.awt.*;
import java.awt.geom.*;

/**
 * Class for the Room object.
 *
 * The Room object is a struct-like construct that is arguably the most
 * important object in Mapwad. Simply stated, a Room is a circular connection
 * of Walls that is singular (that is, Walls cannot cross). These Walls are
 * stored in a special implicit Walls list that is associated with each Room
 * object.
 *
 * @author Avrum Tilman, Josh Weinberg
 * @version CVS $Id: MWRoom.java,v 1.28 2003/12/18 19:32:38 avrum Exp $
 */

public class MWRoom extends MWDataType
{
```

```
// Programmer read-only implicit variable
private MWWalls m_Walls;

//Programmer invisible variable (for GUI)
private Color color;

/**
 * Constructor for the Room Object.
 */
public MWRoom()
{
   m_Walls = new MWWalls(this);
   color = null;
}

/**
 * Get function for the walls member variable.
 *
 * @return The walls object of the room.
 */
public MWWalls getWalls()
{
   return m_Walls;
}

/**
 * Automatically closes the Room.
 *
 * If the Room is already closed nothing happens (ie returns true),
 * otherwise another wall is added to the list that connects the
 * endpoint with the starting point. This method can only be called on
 * a Room with at least 3 walls.
 *
 * @return Whether or not the closure was successful (returns true if
 * the Room was already closed). The value is returned as an MWBoolean.
 */
public MWBoolean close()
{
   int numWalls = m_Walls.getNumWalls();

   if(numWalls < 3)
   {
       return new MWBoolean(false);
   }

   if(isClosed().getValue())
   {
       return new MWBoolean(true);
   }

   float offset = 0;
   float angleToNext = 180;
   float normAngle = 0;
   float x = 0, y = 0;

   for(int i=0; i<numWalls; i++)
   {
       MWWall temp = m_Walls.getWall(i);

       // get length of wall
       float length = temp.getLength().getValue();

       // compute normalized angle
       normAngle = (angleToNext + offset) % 360;

       // compute new point
       if(normAngle > 90 && normAngle < 270)
       {
          x += Math.abs(length*Math.cos(Math.toRadians(normAngle)));
       }
       else
```

```
        {
           x -= Math.abs(length*Math.cos(Math.toRadians(normAngle)));
        }

        if(normAngle < 180)
        {
           y += Math.abs(length*Math.sin(Math.toRadians(normAngle)));
        }
        else
        {
           y -= Math.abs(length*Math.sin(Math.toRadians(normAngle)));
        }

        // update offset and get angle to next wall
        offset += angleToNext+180;
        offset %= 360;
        angleToNext = temp.getAngleToNext().getValue() % 360;
    }

    float angle = 0;
    float hyp = (float)Point2D.distance(x,y,0,0);

    if(y>0)
    {
        if(x>0)
        {
           angle = 360 - (float)Math.toDegrees(Math.asin(y/hyp));
        }
        else
        {
           angle = 180 + (float)Math.toDegrees(Math.asin(y/hyp));
        }
    }
    else
    {
        if(x>0)
        {
           angle = (float)Math.toDegrees(Math.asin(Math.abs(y)/hyp));
        }
        else
        {
           angle = 180 - (float)Math.toDegrees(Math.asin(Math.abs(y)/hyp));
        }
    }

    if(angle > offset)
    {
        angle -= offset;
    }
    else
    {
        angle += (360-offset);
    }

    m_Walls.getWall(numWalls-1).setAngleToNext(new MWFloat(angle));

    MWString param1 = new MWString();
    param1.setValue(MWWall.D_NAME);
    MWString param2 = new MWString();
    param2.setValue(MWWall.D_TEXTURE);

    m_Walls.add(new MWWall(new MWFloat(hyp),
                           new MWFloat(0),
                           param1,
                           param2));
    return new MWBoolean(true);
}

/**
 * Determines whether or not the Room is closed.
 *
```

```
     * Because lengths and angles are floating point values
     * closure will usually not be known with 100% accuracy. Therefore,
     * closure will be assumed if the endpoint of the last wall
     * is within .001 of the startpoint of the first wall. This method
     * requires that there are at least 3 walls in the Room.
     *
     * @return Whether or not the Room is closed. This value is returned as
     * an MWBoolean.
     */
    public MWBoolean isClosed()
    {
        if(m_Walls.getNumWalls() < 3)
        {
            return new MWBoolean(false);
        }

        // distance that is allowed from starting to endpoint
        // and still be considered closed
        float closureError = .001f;
        setCoords();

        if(m_Walls.getWall(0).getStartCoord().distance(m_Walls.getWall(m_Walls.getNumWalls
()-1).getEndCoord()) < closureError)
        {
            return new MWBoolean(true);
        }
        return new MWBoolean(false);
    }

    /**
     * Sets the coordinates for the start and end point of every wall in the
     * room. This function has package level access.
     */
    void setCoords()
    {
        int numWalls = m_Walls.getNumWalls();

        float offset = 0;
        float angleToNext = 180;
        float angle = 0;
        float x = 0,y = 0;

        // make initial point at 0,0
        m_Walls.getWall(0).setStartCoord(x,y);

        for(int i=0; i<numWalls; i++)
        {
            MWWall temp = m_Walls.getWall(i);

            // get length of wall
            float length = temp.getLength().getValue()*MWMap.SCALE;

            // compute normalized angle
            angle = (angleToNext + offset) % 360;

            // compute new point
            if(angle > 90 && angle < 270)
            {
                x += Math.abs(length*Math.cos(Math.toRadians(angle)));
            }
            else
            {
                x -= Math.abs(length*Math.cos(Math.toRadians(angle)));
            }

            if(angle < 180)
            {
                y += Math.abs(length*Math.sin(Math.toRadians(angle)));
            }
            else
            {
```

```
            y -= Math.abs(length*Math.sin(Math.toRadians(angle)));
        }

        // add this point
        temp.setEndCoord(x,y);

        if(i+1<numWalls)
        {
            m_Walls.getWall(i+1).setStartCoord(x,y);
        }

        // update offset and get angle to next wall
        offset += angleToNext+180;
        offset %= 360;
        angleToNext = temp.getAngleToNext().getValue() % 360;
    }
}

/**
 * Fixes the Walls array so that all the walls are the correct size for
 * doors. This method has package level access.
 */
void fixWalls()
{
    for(int i=0;i<m_Walls.getNumWalls();i++)
    {
        if(m_Walls.getWall(i).getIsEntry().getValue())
        {
            fixWall(i);
        }
    }
}

/**
 * Internal function to fix the size of a given wall.
 *
 * This function looks at the specified wall and if it is en entry,
 * breaks up the wall into multiple walls based on the length of the
 * entry and the position of the entry. Basically, it creates new walls
 * one of which is the exact size of the entry (this one will be marked
 * as an entry while the others will be cleared).
 *
 * @param num The index of the wall to fix.
 */
private void fixWall(int num)
{
    MWWall a = m_Walls.getWall(num);

    if(a.getIsEntry().getValue())
    {
        MWWall b = a.getConnectedTo();

        float la, lb;

        la = a.getLength().getValue();
        lb = b.getLength().getValue();
        String pos=a.getConnectionPos().getValue();

        if(la < lb)
        {
            if(pos.equals("center"))
            {
                MWString param1 = new MWString();
                param1.setValue(MWWall.D_NAME);
                MWString param2 = new MWString();
                param2.setValue(b.getTexture().getValue());

                MWWall newBN = new MWWall(new MWFloat((lb-la)/2),
                                          new MWFloat(b.getAngleToNext().getValue()),
                                          param1,
                                          param2);
```

```
                    param1 = new MWString();
                    param1.setValue(MWWall.D_NAME);
                    param2 = new MWString();
                    param2.setValue(b.getTexture().getValue());

                    MWWall newBP = new MWWall(new MWFloat((lb-la)/2),
                                             new MWFloat(180),
                                             param1,
                                             param2);
                    b.setAngleToNext(new MWFloat(180));
                    b.setLength(new MWFloat(la));
                    b.getFromRoom().getWalls().addAfter(b,newBN);
                    b.getFromRoom().getWalls().addAfter(b.getPrevWall(),newBP);
                }
                else if(pos.equals("start"))
                {
                    MWString param1 = new MWString();
                    param1.setValue(MWWall.D_NAME);
                    MWString param2 = new MWString();
                    param2.setValue(b.getTexture().getValue());

                    MWWall newBS = new MWWall(new MWFloat(lb-la),new MWFloat(180),
                                             param1,param2);
                    b.setLength(new MWFloat(la));
                    b.getFromRoom().getWalls().addAfter(b.getPrevWall(),newBS);
                }
                else if(pos.equals("end"))
                {
                    MWString param1 = new MWString();
                    param1.setValue(MWWall.D_NAME);
                    MWString param2 = new MWString();
                    param2.setValue(b.getTexture().getValue());
                    MWWall newBE = new MWWall(new MWFloat(lb-la),
                                             new MWFloat(b.getAngleToNext().getValue()),
                                             param1,param2);
                    b.setAngleToNext(new MWFloat(180));
                    b.setLength(new MWFloat(la));
                    b.getFromRoom().getWalls().addAfter(b,newBE);
                }
            }
            else
            {
                if(la > lb)
                {
                    if (pos.equals("center"))
                    {
                        MWString param1 = new MWString();
                        param1.setValue(MWWall.D_NAME);
                        MWString param2 = new MWString();
                        param2.setValue(a.getTexture().getValue());

                        MWWall newAN = new MWWall(new MWFloat((la-lb)/2),
                                                 new
MWFloat(a.getAngleToNext().getValue()),
                                                 param1,
                                                 param2);

                        param1 = new MWString();
                        param1.setValue(MWWall.D_NAME);
                        param2 = new MWString();
                        param2.setValue(a.getTexture().getValue());

                        MWWall newAP = new MWWall(new MWFloat((la-lb)/2),
                                                 new MWFloat(180),
                                                 param1,
                                                 param2);
                        a.setAngleToNext(new MWFloat(180));
                        a.setLength(new MWFloat(lb));
                        a.getFromRoom().getWalls().addAfter(a,newAN);
                        a.getFromRoom().getWalls().addAfter(a.getPrevWall(),newAP);
                    }
                    else if(pos.equals("start"))
                    {
                        MWString param1 = new MWString();
                        param1.setValue(MWWall.D_NAME);
                        MWString param2 = new MWString();
                        param2.setValue(a.getTexture().getValue());

                        MWWall newAE = new MWWall(new MWFloat(la-lb),
                                                  new
MWFloat(a.getAngleToNext().getValue()),
                                                  param1,param2);
                        a.setAngleToNext(new MWFloat(180));
                        a.setLength(new MWFloat(lb));
                        a.getFromRoom().getWalls().addAfter(a,newAE);
                    }
                    else if(pos.equals("end"))
                    {
                        MWString param1 = new MWString();
                        param1.setValue(MWWall.D_NAME);
                        MWString param2 = new MWString();
                        param2.setValue(a.getTexture().getValue());

                        MWWall newAS = new MWWall(new MWFloat(la-lb),
                                                  new MWFloat(180),
                                                  param1, param2);
                        a.setLength(new MWFloat(lb));
                        a.getFromRoom().getWalls().addAfter(a.getPrevWall(),newAS);
                    }
                }
            }
        }
    }

    /**
     * Centers the room at 0,0 with respect to the given wall's start point
     * or end point. This method has package level access.
     *
     * @param wall The wall that you are centering around.
     * @param startPoint Whether or not the start point is being centered.
     * If true, the start point of the given wall will be moved to 0,0.
     * If false, the end point of the given wall will be moved to 0,0.
     */
    void recenterRoom(MWWall wall, boolean startPoint)
    {
        // make sure that this wall is from this room
        if(!(wall.getFromRoom()).equals(this))
        {
            return;
        }

        float offX = 0,offY = 0;

        if(startPoint)
        {
            offX = (float)(wall.getStartCoord()).getX();
            offY = (float)(wall.getStartCoord()).getY();
        }
        else
        {
            offX = (float)(wall.getEndCoord()).getX();
            offY = (float)(wall.getEndCoord()).getY();
        }

        // translate the walls by the given offsets
        MWTransform.translate(m_Walls,offX,offY);
    }

    /**
     * This function will give you an index of a wall with an entry.
     *
```

```
 * If there are no entry's in the room -1 is returned.  If the
 * room only has entries to itself then 0 is returned.  Otherwise
 * the index of a valid entry wall is returned. This method has package
 * level access.
 *
 * @return The index of an entry wall.
 */
int getEntryWall()
{
   boolean selfLoop = false;
   int size = m_Walls.getNumWalls();

   for(int i=0; i<size; i++)
   {
       MWWall check = m_Walls.getWall(i);

       if(check.getIsEntry().getValue())
       {
           if(check.getFromRoom().equals(check.getConnectedTo().getFromRoom()))
           {
               selfLoop=true;
           }
           else
           {
               return i;
           }
       }
   }
   if(selfLoop)
   {
       return -1;
   }
   return -2;
}

/**
 * Sets the color of this room
 *
 * @param color The color to give to this room.
 */
void setColor(Color color)
{
   this.color = color;
}

/**
 * Returns the color of this room.
 *
 * @return The color of this room.
 */
Color getColor()
{
   return color;
}

/*
 * The following functions are used during the preprocessing phase.
 */

/**
 * Returns the type of data type.
 *
 * @return The type of data type.
 */
public String typeName()
{
   return "Room";
}

/**
 * Whether or not the specified data type is compatible with
```

```
     * an MWRoom.
     *
     * @param type The data type in question.
     * @return Whether or not the specified type is compatible.
     */
    public boolean isCompatibleWith(MWDataType type)
    {
       if(type instanceof MWRoom)
       {
           return true;
       }

       return false;
    }

    /**
     * Returns the member variables associated with this object.
     *
     * @param memberType The name of the member variable to be returned.
     * @return The member variable desired.
     */
    public MWDataType getMember(String memberType)
    {
       if(memberType.equals("Walls"))
       {
           // make this object read only to the user
           m_Walls.setReadOnly();
           return m_Walls;
       }
       return null;
    }
}


******************************
MWThing.java
******************************

package internal;

/**
 * Class for the Thing object.
 *
 * Things are simple entities suppported by Quake that can be put within
 * Rooms. Things include monsters, lava pits, guns, power-ups, etc. A Thing
 * is placed into a Room using a Location.
 *
 * @author Josh Weinberg
 * @version CVS $Id: MWThing.java,v 1.10 2003/12/18 19:32:38 avrum Exp $
 */

public class MWThing extends MWDataType
{
    /**
     * Default class variable values.
     */
    static final String D_TYPE = "";
    static final MWLocation D_POSITION = null;

    /**
     * Thing member variables.
     */
    // programmer read-write variables
    private MWString m_Type;
    private MWReferenceType m_Position;

   /**
     * Constructor for a Thing reference.
     *
     *  @param name The Mapwad variable name
     */
```

```
public MWThing(String name)
{
   super(name);
}

/**
 * Constructor for the Thing object.
 *
 * Empty constructor that initializes member variables to default values.
 */
public MWThing()
{
   MWString temp = new MWString();
   temp.setValue(D_TYPE);
   m_Type = temp;
   m_Position = new MWReferenceType(new MWLocation());
}

/**
 * Constructor for the Thing object.
 *
 * @param type The type of the Thing.
 * @param position The position of the Thing.
 */
public MWThing(MWString type, MWLocation position)
{
   m_Type = type;
   m_Position.setObject(position);
}

/**
 * Sets the type of the Thing.
 *
 * @param type The type of the Thing.
 */
public void setType(MWString type)
{
   m_Type = type;
}

/**
 * Set the position of the Thing.
 *
 * @param position The position of the Thing.
 */
public void setPosition(MWLocation position)
{
   m_Position.setObject(position);
}

/**
 * Returns the type of the Thing.
 *
 * @return The type of the Thing.
 */
public MWString getType()
{
   return m_Type;
}

/**
 * Returns the position of the Thing.
 *
 * @return The position of the Thing.
 */
public MWLocation getPosition()
{
   return (MWLocation)m_Position.getObject();
}

/*
```

```
     * The following functions are used during the preprocessing phase.
     */

    /**
     * Returns the type of data type.
     *
     * @return The type of data type.
     */
    public String typeName()
    {
       return "Thing";
    }

    /**
     * Whether or not the specified data type is compatible with
     * an MWThing.
     *
     * @param type The data type in question.
     * @return Whether or not the specified type is compatible.
     */
    public boolean isCompatibleWith(MWDataType type)
    {
       if(type instanceof MWThing)
           return true;

       return false;
    }

    /**
     * Returns the member variables associated with this object.
     *
     * @param memberType The type of member variable to be returned.
     * @return The member variable desired.
     */
    public MWDataType getMember(String memberType)
    {
       if(memberType.equals("Type"))
       {
           return m_Type;
       }
       if(memberType.equals("Position"))
       {
           return m_Position;
       }

       return null;
    }
}


******************************
MWWall.java
******************************

package internal;

import java.awt.geom.*;

/**
 * Class for the Wall object.
 *
 * Walls are a struct-like construct that are used to make up Rooms. Walls
 * are made up of implicit variables that can be set and changed depending
 * on the desired situation. Each of the Wall's implicit variables has a
 * default value.
 *
 * @author Avrum Tilman, Josh Weinberg
 * @version CVS $Id: MWWall.java,v 1.38 2003/12/18 19:32:39 avrum Exp $
 */

public class MWWall extends MWDataType
```

```
{
    /**
     * Default class variable values.
     */
    public static final float D_LENGTH = 20;
    public static final float D_ANGLE_TO_NEXT = 90;
    public static final String D_NAME = "";
    public static final String D_TEXTURE = "";

    /**
     * Wall member variables.
     */
    // programmer read-write variables
    private MWFloat m_Length;
    private MWFloat m_AngleToNext;
    private MWString m_Name;
    private MWString m_Texture;
    private MWString m_ConnectionPos;

    // programmer read-only variables
    private MWBoolean m_IsEntry;
    private MWWall m_ConnectedTo;
    private MWRoom m_FromRoom;

    // programmer invisible variables
    private MWRoom m_origRoom;
    private Point2D.Float m_StartCoord;
    private Point2D.Float m_EndCoord;


   /**
     * Constructor for a Wall reference.
     * When a variable is constructed with a name, it has not been
     * instantiated yet, so we don't need to do any intialization here.
     *
     *  @param name The Mapwad variable name
     */
    public MWWall(String name)
    {
       super(name);
    }

    /**
     * Constructor for the Wall object.
     *
     * Empty constructor that initializes member variables to default values.
     */
    public MWWall()
    {
       m_Length = new MWFloat(D_LENGTH);
       m_AngleToNext = new MWFloat(D_ANGLE_TO_NEXT);
       m_Name = new MWString();
       m_Name.setValue(D_NAME);
       m_Texture = new MWString();
       m_Texture.setValue(D_TEXTURE);
       m_ConnectedTo = null;
       m_FromRoom = null;
       m_origRoom = null;
       m_IsEntry = new MWBoolean(false);
       m_ConnectionPos = new MWString();

       m_StartCoord = null;
       m_EndCoord = null;
    }

    /**
     * Constructor for the Wall object.
     *
     * @param length The length of this Wall.
     * @param angleToNext The angle between this Wall and the next Wall.
     * @param name The name of this Wall.
     * @param texture The texture to be mapped to this Wall.
     */
    public MWWall(MWFloat length, MWFloat angleToNext, MWString name, MWString texture)
    {
       m_Length = length;
       m_AngleToNext = angleToNext;
       m_Name = name;
       m_Texture = texture;

       m_IsEntry = new MWBoolean(false);
       m_ConnectedTo = null;
       m_FromRoom = null;
       m_origRoom = null;
       m_ConnectionPos = new MWString();

       m_StartCoord=null;
       m_EndCoord=null;
    }

    /**
     * Constructor for the Wall object.
     *
     * @param length The length of this Wall.
     * @param angleToNext The angle between this Wall and the next Wall.
     * @param name The name of this Wall.
     */
    public MWWall(MWFloat length, MWFloat angleToNext, MWString name)
    {
       m_Length = length;
       m_AngleToNext = angleToNext;
       m_Name = name;

       m_Texture = new MWString();
       m_Texture.setValue(D_TEXTURE);

       m_IsEntry = new MWBoolean(false);
       m_ConnectedTo = null;
       m_FromRoom = null;
       m_origRoom = null;
       m_ConnectionPos = new MWString();

       m_StartCoord = null;
       m_EndCoord = null;
    }

    /**
     * Constructor for the Wall object.
     *
     * @param length The length of this Wall.
     * @param angleToNext The angle between this Wall and the next Wall.
     */
    public MWWall(MWFloat length, MWFloat angleToNext)
    {
       m_Length = length;
       m_AngleToNext = angleToNext;

       m_Name = new MWString();
       m_Name.setValue(D_NAME);
       m_Texture = new MWString();
       m_Texture.setValue(D_TEXTURE);

       m_IsEntry = new MWBoolean(false);
       m_ConnectedTo = null;
       m_FromRoom = null;
       m_origRoom = null;
       m_ConnectionPos = new MWString();

       m_StartCoord = null;
       m_EndCoord = null;
    }
```

```
/**
 * Constructor for the Wall object.
 *
 * @param length The length of this Wall.
 */
public MWWall(MWFloat length)
{
   m_Length = length;
   new MWFloat(D_ANGLE_TO_NEXT);
   m_Name = new MWString();
   m_Name.setValue(D_NAME);
   m_Texture = new MWString();
   m_Texture.setValue(D_TEXTURE);

   m_IsEntry = new MWBoolean(false);
   m_ConnectedTo = null;
   m_FromRoom = null;
   m_origRoom = null;
   m_ConnectionPos = new MWString();

   m_StartCoord = null;
   m_EndCoord = null;
}

/**
 * Sets the length of the Wall.
 *
 * @param length The length of this Wall.
 */
public void setLength(MWFloat length)
{
   m_Length = length;
}

/**
 * Sets the angle between this Wall and the next Wall.
 *
 * @param angleToNext The angle between this Wall and the next Wall.
 */
public void setAngleToNext(MWFloat angleToNext)
{
   m_AngleToNext = angleToNext;
}

/**
 * Sets the name of this Wall.
 *
 * @param name The name of this Wall.
 */
public void setName(MWString name)
{
   if(m_FromRoom != null)
   {
       if(m_FromRoom.getWalls().getWall(m_Name)==this)
       {
          m_FromRoom.getWalls().getHash().remove(m_Name.getValue());
       }
       m_FromRoom.getWalls().getHash().put(name.getValue(),this);
   }
   m_Name = name;
}
/**
 * Sets the texture of this Wall.
 *
 * @param texture The texture to be mapped to this Wall.
 */
public void setTexture(MWString texture)
{
   m_Texture = texture;
}
```

```
/**
 * Indicates whether or not this Wall is an entry.
 *
 * A Wall is considered an entry if it is attached to another Wall.
 * @param isEntry Whether or not this Wall is an entry.
 */
void setIsEntry(MWBoolean isEntry)
{
   m_IsEntry = isEntry;
}

/**
 * Sets the Wall that this Wall is connected to.
 *
 * @param connectedTo The Wall that this Wall is connected to.
 */
void setConnectedTo(MWWall connectedTo)
{
   m_ConnectedTo = connectedTo;
}

/**
 * Sets the Room that contains this Wall.
 *
 * @param fromRoom The Room that contains this Wall.
 */
void setFromRoom(MWRoom fromRoom)
{
   m_FromRoom = fromRoom;
}

/**
 * Sets the original Room that contained this Wall.
 *
 * @param origRoom The original Room that contained this Wall.
 */
void setOrigRoom(MWRoom origRoom)
{
   m_origRoom = origRoom;
}

/**
 * Sets the connected position of this Wall.
 *
 * Possible values are left, right, and center. These positions
 * refer to the the edge of this Wall that is connected to another
 * Wall.
 * @param connectionPos The connected position of this Wall.
 */
void setConnectionPos(MWString connectionPos)
{
   connectionPos.setValue(connectionPos.getValue().toLowerCase());
   m_ConnectionPos = connectionPos;
}

/**
 * This method updates a wall's connection position and updates the wall
 * it is connected to.
 *
 * @param pos The position of the new connection point
 */
public void updateConnectionPos(MWString pos)
{
   if (m_IsEntry.getValue())
   {
       String p = pos.getValue().toLowerCase();
       m_ConnectionPos=new MWString(p);
       if (p.equals("center"))
       {
          MWString temp = new MWString();
          temp.setValue("center");
```

```
            m_ConnectedTo.setConnectionPos(temp);
        }
        else if (p.equals("start"))
        {
            MWString temp = new MWString();
            temp.setValue("end");
            m_ConnectedTo.setConnectionPos(temp);
        }
        else if (p.equals("end"))
        {
            MWString temp = new MWString();
            temp.setValue("start");
            m_ConnectedTo.setConnectionPos(temp);
        }
    }
}

/**
 * Sets the start coordinate of the wall.
 *
 * @param x The x coordinate
 * @param y The y coordinate
 */
void setStartCoord(float x, float y)
{
    m_StartCoord = new Point2D.Float(x,y);
}

/**
 * Sets the end coordinate of the wall.
 *
 * @param x The x coordinate
 * @param y The y coordinate
 */
void setEndCoord(float x, float y)
{
    m_EndCoord = new Point2D.Float(x,y);
}

/**
 * Returns the value of this Wall's length
 *
 * @return The length of this Wall.
 */
public MWFloat getLength()
{
    return m_Length;
}

/**
 * Returns the value of the angle between this Wall and the next Wall.
 *
 * @return The angle between this Wall and the next Wall.
 */
public MWFloat getAngleToNext()
{
    return m_AngleToNext;
}

/**
 * Returns the name associated with this Wall.
 *
 * @return The name of this Wall.
 */
public MWString getName()
{
    return m_Name;
}

/**
 * Returns the name of the texture to be mapped onto this Wall.
 *
 * @return The name of the texture to be mapped onto this Wall.
 */
public MWString getTexture()
{
    return m_Texture;
}

/**
 * Returns the Wall connected "in front" of this Wall.
 *
 * Based on this Wall's associated room, the next Wall, relative to this
 * Wall, in the room's Walls list will be returned.
 *
 * @return The next Wall in this Wall's room's Walls list.
 */
public MWWall getNextWall()
{
    if(m_FromRoom != null)
    {
        return m_FromRoom.getWalls().getNext(this);
    }
    return null;
}

/**
 * Returns the wall connected "behind" this Wall.
 *
 * Based on this Wall's associated room, the previous wall, relative to
 * this Wall, in the room's Walls list will be returned.
 *
 * @return The previous Wall in this Wall's room's Walls list.
 */
public MWWall getPrevWall()
{
    if(m_FromRoom != null)
    {
        return m_FromRoom.getWalls().getPrev(this);
    }
    return null;
}

/**
 * Returns whether or not this Wall is an entry.
 *
 * @return Whether or not this Wall is an entry.
 */
public MWBoolean getIsEntry()
{
    return m_IsEntry;
}

/**
 * Returns the Wall that this Wall is connected to.
 *
 * If this Wall is not an entry, the m_ConnectedTo variable will be
 * set to null, so this function will return null.
 * @return The Wall that this Wall is connected to.
 */
public MWWall getConnectedTo()
{
    return m_ConnectedTo;
}

/**
 * Returns the Room that contains this Wall.
 *
 * @return The Room that contains this Wall.
 */
public MWRoom getFromRoom()
{
    return m_FromRoom;
}

/**
 * Returns the original Room that contained this Wall.
 *
 * @return The original Room that contained this Wall.
 */
MWRoom getOrigRoom()
{
    return m_origRoom;
}

/**
 * Returns the connected position of the Wall.
 *
 * If this Wall is not connected, it will return null.
 * @return The connected position of this Wall.
 */
public MWString getConnectionPos()
{
    return m_ConnectionPos;
}

/**
 * Returns the starting coordinate of the wall
 *
 * @return The starting coordinate of the wall
 */
Point2D.Float getStartCoord()
{
    return m_StartCoord;
}

/**
 * Returns the ending coordinate of the wall
 *
 * @return The ending coordinate of the wall
 */
Point2D.Float getEndCoord()
{
    return m_EndCoord;
}

/**
 * Attaches this Wall to the specified Wall at the specified
 * position.
 *
 * @param b the other wall to attach to
 * @param pos the position where the walls are attached
 * @return Whether or not the attach succeeded.
 */
public MWBoolean attach(MWWall b, MWString pos) throws MWException
{
    String p = pos.getValue().toLowerCase();
    if(!(p.equals("center")||p.equals("start")||p.equals("end")))
        throw new MWException("Illegal position value");

    if(b==null)
        return new MWBoolean(false);

    if(b.isNull())
        return new MWBoolean(false);
    //check to see if these Walls are in the same Room
    if(m_FromRoom == b.getFromRoom() && m_FromRoom != null)
    {
        return new MWBoolean(false);
    }

    // check to see if these Walls are already attached
    if(m_IsEntry.getValue() || b.getIsEntry().getValue())
    {
        return new MWBoolean(false);
    }

    // update this Wall's attributes
    m_IsEntry = new MWBoolean(true);
    m_ConnectedTo = b;
    m_ConnectionPos = new MWString();
    m_ConnectionPos.setValue(p);

    // update the specified Wall's attributes
    b.setIsEntry(new MWBoolean(true));
    b.setConnectedTo(this);
    if (p.equals("center"))
    {
        MWString temp = new MWString();
        temp.setValue("center");
        b.setConnectionPos(temp);
    }
    else if(p.equals("start"))
    {
        MWString temp = new MWString();
        temp.setValue("end");
        b.setConnectionPos(temp);
    }
    else
    {
        MWString temp = new MWString();
        temp.setValue("start");
        b.setConnectionPos(temp);
    }


    return new MWBoolean(true);
}

/**
 * Aligns the wall (and therefore its associated room) with the
 * y-axis and the "outside" of the wall facing the positive x-direction.
 *
 * Aligns the wall (and room) so the the given wall's start point
 * (or end point - if attachee) is at 0,0 and the end point
 * (or start point - if attachee) lies along the positive y-axis.
 * Furthermore, if the "outside" of the wall is facing in the
 * negative x-direction (or positive x-direction - if attachee), the
 * room is reflected across the y-axis. The attachee is the room that is
 * being attached to another room (ie it's walls will be inserted into the
 * other room).
 * @param attacher Whether or not this is the room doing the attach (ie it
 * will absorb the walls of the attachee).
 */
public void align(boolean attacher)
{
    // create error value for "close" point
    float error = .001f;

    // make sure that this wall belongs to a room
    if(m_FromRoom == null)
    {
        return;
    }

    GeneralPath path = null;
    float rotationAngle = 0;
    // see description of attacher vs attachee above
    if(attacher)
    {
        // recenter the start point of this wall
        m_FromRoom.recenterRoom(this, true);
        rotationAngle = MWTransform.getRotationAngle((float)m_EndCoord.getX(),
                                                     (float)m_EndCoord.getY());
        MWTransform.rotate(m_FromRoom.getWalls(),rotationAngle);

        // convert to path and get winding information
        // by checking if the point (.001,midpoint of wall) is contained
        // in the path. If it is, the room must be reflected
        // across the y-axis.
        path = MWTransform.convertToPath(m_FromRoom.getWalls());
        float midpoint = (float)m_EndCoord.getY()/2;

        if(path.contains(error,midpoint))
        {
           MWTransform.reflectY(m_FromRoom.getWalls());
        }
    }
    else
    {
        // recenter the end point of this wall
        m_FromRoom.recenterRoom(this, false);
        rotationAngle = MWTransform.getRotationAngle((float)m_StartCoord.getX(),
                                                   (float)m_StartCoord.getY());
        MWTransform.rotate(m_FromRoom.getWalls(),rotationAngle);

        // convert to path and get winding information
        // by checking if the point (-.001,midpoint of wall) is contained
        // in the path. If it is, the room must be reflected
        // across the y-axis.
        path = MWTransform.convertToPath(m_FromRoom.getWalls());
        float midpoint = (float)m_StartCoord.getY()/2;

        if(path.contains((-1*error),midpoint))
        {
           MWTransform.reflectY(m_FromRoom.getWalls());
        }
    }
}

/**
 * This function merges 2 rooms into one new room.
 *
 * The room that this room is connected to is added to the room that this
 * wall is currently in.  The new room no longer contains either door
 * wall.  If this wall is not a door then nothing happens
 */
public void merge()
{
    if (m_IsEntry.getValue())
    {
        if(m_ConnectedTo!=null)
        {
           MWWalls walls=m_FromRoom.getWalls();
           MWWall prev=getPrevWall();
           MWWall door=m_ConnectedTo;
           MWWall cur=door.getNextWall();
           while(cur!=door)
           {
               MWWall next=cur.getNextWall();

               cur.setFromRoom(m_FromRoom);
               walls.addAfter(prev,cur);
               prev=cur;
               cur=next;
           }
           walls.remove(this);
        }
    }
}

/**
 * Is this wall near wall n
 *
 * @param n the wall to check
```

```
     * @return Whether or not the this wall is near the specified wall.
     */
    boolean isNear(MWWall n)
    {
        float err=.001f;
        if(m_StartCoord.distance(n.getEndCoord())<err &&
m_EndCoord.distance(n.getStartCoord())<err)
            return true;
        return false;
    }

    /**
     * Overriden toString method.
     *
     * This method prints out the following information about this wall:
     * 1) Name (if applicable)
     * 2) Angle to next wall
     * 3) Start coordinate
     * 4) End coordinate
     *
     * @return The string representation of the wall.
     */
    public String toString()
    {
        String message = "";

        message += "Name: " + m_Name.getValue() + "\n";
        message += "Length: " + m_Length.getValue() + "\n";
        message += "Angle to Next: " + m_AngleToNext.getValue() + "\n";
        message += "Texture: " + m_Texture.getValue() + "\n";
        message += "Entry: " + m_IsEntry.getValue() + "\n";

        if(m_StartCoord != null)
        {
            message += "Start Coordinate: " + m_StartCoord + "\n";
        }
        else
        {
            message += "Start Coordinate: null\n";
        }
        if(m_EndCoord != null)
        {
            message += "End Coordinate: " + m_EndCoord + "\n";
        }
        else
        {
            message += "End Coordinate: null\n";
        }
        return message;
    }

    /*
     * The following functions are used during the preprocessing phase.
     */

    /**
     * Returns the type of data type.
     *
     * @return The type of data type.
     */
    public String typeName()
    {
        return "Wall";
    }

    /**
     * Whether or not the specified data type is compatible with
     * an MWWall.
     *
     * @param type The data type in question.
     * @return Whether or not the specified type is compatible.
```

```
 */
public boolean isCompatibleWith(MWDataType type)
{
   if(type instanceof MWWall)
       return true;

   return false;
}

/**
 * Returns the member variables associated with this object.
 *
 * @param memberType The type of member variable to be returned.
 * @return The member variable desired.
 */
public MWDataType getMember(String memberType)
{
   if(memberType.equals("Length"))
   {
       return m_Length;
   }
   if(memberType.equals("AngleToNext"))
   {
       return m_AngleToNext;
   }
   if(memberType.equals("Name"))
   {
       //so we know to update the hashtable
       if(m_origRoom != null)
          m_origRoom.getWalls().namesChanged();

       return m_Name;
   }
   if(memberType.equals("Texture"))
   {
       return m_Texture;
   }
   if(memberType.equals("Next"))
   {
       MWWall temp = getNextWall();

       // return the next wall in the list
       // if it is null, return this wall.
       if(temp == null)
       {
          temp=this;
       }

       MWReferenceType reftype = new MWReferenceType(temp);
       reftype.setReadOnly();
       return reftype;
   }
   if(memberType.equals("Prev"))
   {
       MWWall temp = getPrevWall();

       // return the next wall in the list
       // if it is null, return this wall.
       if(temp == null)
       {
          temp=this;
       }

       MWReferenceType reftype = new MWReferenceType(temp);
       reftype.setReadOnly();
       return reftype;
   }
   if(memberType.equals("IsEntry"))
   {
       // set variable as read-only for programmer
       m_IsEntry.setReadOnly();
```

```
            return m_IsEntry;
        }
        if(memberType.equals("ConnectedTo"))
        {
            // set variable as read-only for programmer
            m_ConnectedTo.setReadOnly();
            return m_ConnectedTo;
        }
        if(memberType.equals("FromRoom"))
        {
            // set variable as read-only for programmer
            m_FromRoom.setReadOnly();
            return m_FromRoom;
        }

        return null;
    }
}


******************************
MWWalls.java
******************************

package internal;

import java.util.*;

/**
 * Class for the Walls LinkedList/Array/Associative Array
 *
 * @author Avrum Tilman
 * @version CVS $Id: MWWalls.java,v 1.22 2003/12/16 04:00:28 bhs16 Exp $
 */
public class MWWalls extends MWDataType
{
    private LinkedList m_Walls;
    private Hashtable m_Hash;
    private MWRoom m_Room;
    private boolean m_namesChanged=false;

    /**
     * Default Constructor for the Walls object.
     *
     * @param room The Room that this Walls object exists in
     */
    MWWalls(MWRoom room)
    {
        m_Room=room;
        m_Walls = new LinkedList();
        m_Hash = new Hashtable();
    }

    /**
     * Returns the element at the specified index.
     *
     * @param index The index of the element desired.
     * @param line The line number that is calling this method.
     * @return The element at the specified index.
     * @throws MWException If the index is out of bounds or the key doesn't
     * exist.
     */
    public MWDataType elementAt(MWDataType index, int line) throws MWException
    {
        MWDataType wall=null;

        if( index instanceof MWInt )
        {
            wall = getWall((MWInt)index);

            if(wall == null)
                throw new MWException(line,"Index out of bounds");
        }
        else if( index instanceof MWString )
        {
            wall = getWall((MWString)index);

            if(wall == null)
                throw new MWException(line,"Key does not exist");
        }
        else
            MWArray.errorWrongIndexType(index,line);

        return new MWReferenceType(wall);
    }

    /**
     * Add a wall to the Walls object.
     *
     * @param newWall The Wall to add to the Walls object.
     * @return Whether the operation succeeded.
     */
    public boolean add(MWWall newWall)
    {
        if (m_Walls.size()==0)
        {
            if(newWall.getFromRoom()==null)
            {
                newWall.setFromRoom(m_Room);
                newWall.setOrigRoom(m_Room);
            }
            else if(newWall.getFromRoom()!=m_Room)
                return false;
            m_Walls.add(newWall);
            m_Hash.put(newWall.getName().getValue(),newWall);
            return true;
        }
        if (newWall.getFromRoom()==null)
        {
            newWall.setFromRoom(m_Room);
            newWall.setOrigRoom(m_Room);
        }
        else if(newWall.getFromRoom()!=m_Room)
                return false;
        m_Walls.add(newWall);
        m_Hash.put(newWall.getName().getValue(),newWall);
        return true;
    }

    /**
     * Add newWall to the Walls object after locat.
     *
     * @param locat The location of where to add the new Wall.
     * @param newWall The Wall to add to the Walls object.
     * @return Whether the operation succeeded.
     */
    public boolean addAfter(MWWall locat, MWWall newWall)
    {
        if(newWall.getFromRoom()==null)
        {
            newWall.setFromRoom(m_Room);
            newWall.setOrigRoom(m_Room);
        }
        else if(newWall.getFromRoom()!=m_Room)
            return false;

        if (m_Walls.size()==0)
        {
            m_Walls.add(newWall);
            m_Hash.put(newWall.getName().getValue(),newWall);
            return true;
        }
        if(locat==null)
        {
            m_Walls.add(newWall);
            m_Hash.put(newWall.getName().getValue(),newWall);
            return true;
        }
        if(locat.getFromRoom()!=m_Room)
            return false;

        m_Walls.add(m_Walls.indexOf(locat)+1,newWall);
        m_Hash.put(newWall.getName().getValue(),newWall);

        return true;
    }

    /**
     * Remove Wall rem from the Walls object
     *
     * @param rem The Wall to remove.
     */
    public void remove(MWWall rem)
    {
        m_Walls.remove(rem);
        m_Hash.remove(rem.getName().getValue());
    }

    /**
     * Remove Wall rem from the Walls object
     *
     * @param index The Wall to remove.
     */
    public void remove(int index)
    {
        m_Hash.remove(((MWWall)m_Walls.get(index)).getName().getValue());
        m_Walls.remove(index);
    }

    /**
     * Remove Wall rem from the Walls object
     *
     * @param index The Wall to remove.
     */
    public void remove(MWInt index)
    {
        m_Hash.remove(((MWWall)m_Walls.get(index.getValue())).getName().getValue());
        m_Walls.remove(index);
    }

    /**
     * Get the Wall with the given index
     *
     * @param index The index in the array of the wall to get.
     * @return The requested wall.
     */
    public MWWall getWall(int index)
    {
        if (index<0||index>=m_Walls.size())
            return null;
        return (MWWall)m_Walls.get(index);
    }

    /**
     * Get the Wall with the given index
     *
     * @param index The index in the array of the wall to get.
     * @return The requested wall.
     */
    public MWWall getWall(MWInt index)
    {
        if (index.getValue()<0||index.getValue()>=m_Walls.size())
```

```
        return null;
    return (MWWall)m_Walls.get(index.getValue());
}

/**
 * Get the wall with the given key
 *
 * @param key The key in the associative array of the wall to get.
 * @return The requested wall.
 */
public MWWall getWall(MWString key)
{
    //return (MWWall)m_Hash.get(key.getValue());
    return getWall(key.getValue());
}

/**
 * Get the wall with the given key
 *
 * @param key The key in the associative array of the wall to get.
 * @return The requested wall.
 */
public MWWall getWall(String key)
{
    if(m_namesChanged)
     {
        Iterator itWall = m_Walls.iterator();
        MWWall wall;

        m_Hash.clear();

        while(itWall.hasNext())
        {
            wall = (MWWall)itWall.next();

            m_Hash.put(wall.getName().getValue(),wall);
        }
    }

    return (MWWall)m_Hash.get(key);
}

void namesChanged()
{
    m_namesChanged=true;
}

/**
 * Get the Wall next to cur
 *
 * @param cur The current wall.
 * @return The next wall.
 */
public MWWall getNext(MWWall cur)
{
    int index;
    index=m_Walls.indexOf(cur);
    if (index==-1)
        return null;
    index++;
    ListIterator itr=m_Walls.listIterator(index);
    if (itr.hasNext())
        return (MWWall)itr.next();
    return (MWWall)m_Walls.getFirst();
}

/**
 * Get the Wall previous to cur
 *
 * @param cur The current wall.
 * @return The previous wall.
 */
public MWWall getPrev(MWWall cur)
{
   int index;
   index=m_Walls.indexOf(cur);
   if (index==-1)
       return null;
   ListIterator itr=m_Walls.listIterator(index);
   if (itr.hasPrevious())
       return (MWWall)itr.previous();
   return (MWWall)m_Walls.getLast();
}

/**
 * Returns the number of walls in the list.
 *
 * @return The number of walls in the list.
 */
public int getNumWalls()
{
   return m_Walls.size();
}

/**
 * Returns the room that is associated with this list of walls.
 *
 * @return The room associated with this list of walls.
 */
MWRoom getRoom()
{
   return m_Room;
}

/**
 * Internal method that returns the hashtable for the associative lookup.
 *
 * @return The Hashtable.
 */
Hashtable getHash()
{
   return m_Hash;
}

/**
 * Overriden toString method.
 *
 * This method prints out the list of walls.
 * @return The string representation of the list of walls.
 */
public String toString()
{
   String message = "";

   for(int i=0; i<m_Walls.size(); i++)
   {
       message += ((MWWall)m_Walls.get(i)).toString() + "\n";
   }

   return message;
}

public LinkedList getWalls()
{
   return m_Walls;
}

/*
 * The following functions are used during the preprocessing phase.
 */

/**
```

```
     * Returns the type of data type.
     *
     * @return The type of data type.
     */
    public String typeName()
    {
        return "Walls";
    }

   /**
     * Assignment operator.  Called by MapwadWalker
     *  @param data The data type to be assigned to
     *  @param line The line number where the assignment occured
     */
    public MWDataType assign(MWDataType data, int line) throws MWException
    {
        if( data instanceof MWWalls )
        {
            m_Room=((MWWalls)data).m_Room;
            m_Walls=((MWWalls)data).m_Walls;
            m_Hash=((MWWalls)data).m_Hash;

            m_initialized=true;
            return this;
        }

        return error(data,"=", line);
    }

    /**
     * Returns the member variables associated with this object.
     *
     * @param memberType The type of member variable to be returned.
     * @return The member variable desired.
     */
    public MWDataType getMember(String memberType)
    {
        if(memberType.equals("Size"))
        {
            // get the current size of the walls list
            MWInt size = new MWInt(getNumWalls());

            // set variable as read-only for programmer
            size.setReadOnly();
            return size;
        }

        return null;
    }
}
```

8.1.10. Basic Types

```
******************************
MWBoolean.java
******************************

package internal;

/**
 * Mapwad boolean type.
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWBoolean.java,v 1.10 2003/12/18 19:32:35 avrum Exp $
 */

public class MWBoolean extends MWDataType
{
    boolean m_value;
    public MWBoolean()
    {}

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

    public MWBoolean(boolean value)
    {
       m_value = value;
       m_initialized=true;
    }

    public boolean getValue()
    { return m_value; }

    public String typeName()
    { return "boolean"; }

    public boolean isCompatibleWith(MWDataType type)
    {
       if(type instanceof MWBoolean)
           return true;

       return false;
    }

    public String toString()
    { return Boolean.toString(m_value); }

    public MWDataType assign(MWDataType data,int line) throws MWException
    {
       m_value = ((MWBoolean)data).m_value;
       return this;
    }

    public MWDataType and(MWDataType data, int line) throws MWException
    {
       if( data instanceof MWBoolean )
           return new MWBoolean( m_value && ((MWBoolean)data).m_value);
       else
           return error(data,"&&",line);
    }

    public MWDataType or(MWDataType data, int line) throws MWException
    {
       if( data instanceof MWBoolean )
           return new MWBoolean( m_value || ((MWBoolean)data).m_value);
       else
           return error(data,"||",line);
    }

    public MWDataType not(int line)
    {
       return new MWBoolean(!m_value);
    }
}


******************************
MWFloat.java
******************************

package internal;

/**
 * Mapwad float type.
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWFloat.java,v 1.19 2003/12/18 19:32:36 avrum Exp $
 */

public class MWFloat extends MWDataType
{
    float m_value;

    public MWFloat()
    { }

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

    public MWFloat(float value)
    {
        m_value = value;
        setInitialized();
    }

    public MWFloat(String name, float value)
    {
        super(name);
        m_value = value;
        setInitialized();
    }

    public String typeName()
    { return "float"; }

    public boolean passByValue()
    { return true; }

    public boolean isCompatibleWith(MWDataType type)
    {
        if(type instanceof MWInt || type instanceof MWFloat)
            return true;

        return false;
    }

    public float getValue()
    { return m_value; }

    void setValue(int value)
    { m_value=value; }

    public String toString()
    { return Float.toString(m_value); }


    public MWDataType assign(MWDataType data, int line) throws MWException
    {
        if( data instanceof MWFloat )
        {
            m_value = ((MWFloat)data).m_value;
            return this;
        }
        else if( data instanceof MWInt )
        {
            m_value = ((MWInt)data).m_value;
            return this;
        }

        return error(data,"=",line);
    }

    public MWDataType equals(MWDataType data, int line) throws MWException
    {
```

```
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value == ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value == ((MWFloat)data).m_value);
    }

    return error(data,"==",line);
}

public MWDataType notEq(MWDataType data, int line) throws MWException
{
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value != ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value != ((MWFloat)data).m_value);
    }

    return error(data,"==",line);
}

public MWDataType greaterThan(MWDataType data, int line) throws MWException
{
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value > ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value > ((MWFloat)data).m_value);
    }

    return error(data,">",line);
}

public MWDataType lessThanEq(MWDataType data, int line) throws MWException
{
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value <= ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value <= ((MWFloat)data).m_value);
    }

    return error(data,"<=",line);
}

public MWDataType greaterThanEq(MWDataType data, int line) throws MWException
{
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value >= ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value >= ((MWFloat)data).m_value);
    }

    return error(data,">=",line);
}

public MWDataType lessThan(MWDataType data, int line) throws MWException
{
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value < ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value < ((MWFloat)data).m_value);
    }

    return error(data,"<",line);
}

public MWDataType plusEq(MWDataType data, int line) throws MWException
{
    if( data instanceof MWFloat )
    {
        m_value += ((MWFloat)data).m_value;
        return this;
    }
    if( data instanceof MWInt )
    {
        m_value += (float)((MWInt)data).m_value;
        return this;
    }

    return error(data,"+=",line);
}

public MWDataType minusEq(MWDataType data, int line) throws MWException
{
    if( data instanceof MWFloat )
    {
        m_value -= ((MWFloat)data).m_value;
        return this;
    }
    if( data instanceof MWInt )
    {
        m_value -= (float)((MWInt)data).m_value;
        return this;
    }

    return error(data,"-=",line);
}

public MWDataType timesEq(MWDataType data, int line) throws MWException
{
    if( data instanceof MWFloat )
    {
        m_value *= ((MWFloat)data).m_value;
        return this;
    }
    if( data instanceof MWInt )
    {
        m_value *= (float)((MWInt)data).m_value;
        return this;
    }

    return error(data,"+=",line);
}

public MWDataType divideEq(MWDataType data, int line) throws MWException
{
    if( data instanceof MWFloat )
    {
        m_value /= ((MWFloat)data).m_value;
        return this;
    }
    if( data instanceof MWInt )
    {
        m_value /= (float)((MWInt)data).m_value;
        return this;
    }
    return error(data,"/=",line);
}

public MWDataType modEq(MWDataType data, int line) throws MWException
{
    if( data instanceof MWFloat )
    {
        m_value %= ((MWFloat)data).m_value;
        return this;
    }
    if( data instanceof MWInt )
    {
        m_value %= (float)((MWInt)data).m_value;
        return this;
    }

    return error(data,"%=",line);
}

public MWDataType plusPlus(int line)
{
    m_value++;
    return this;
}

public MWDataType minusMinus(int line)
{
    m_value--;
    return this;
}

public MWDataType plusSign(int line)
{
    return new MWFloat(+m_value);
}

public MWDataType minusSign(int line)
{
    return new MWFloat(-m_value);
}

public MWDataType add(MWDataType data, int line) throws MWException
{
    if( data instanceof MWFloat )
        return new MWFloat( m_value + ((MWFloat)data).m_value );
    if( data instanceof MWInt )
        return new MWFloat( m_value + (float)((MWInt)data).m_value );
    else
        return error(data,"+",line);
}

public MWDataType subtract(MWDataType data, int line) throws MWException
{
    if( data instanceof MWFloat )
        return new MWFloat( m_value - ((MWFloat)data).m_value );
    if( data instanceof MWInt )
        return new MWFloat( m_value - (float)((MWInt)data).m_value );
    else
        return error(data,"-",line);
}

public MWDataType multiply(MWDataType data, int line) throws MWException
{
    if( data instanceof MWFloat )
        return new MWFloat( m_value * ((MWFloat)data).m_value );
    if( data instanceof MWInt )
        return new MWFloat( m_value * (float)((MWInt)data).m_value );
    else
        return error(data,"/",line);
}
```

```
    public MWDataType divide(MWDataType data, int line) throws MWException
    {
       if( data instanceof MWFloat )
           return new MWFloat( m_value / ((MWFloat)data).m_value );
       if( data instanceof MWInt )
           return new MWFloat( m_value / (float)((MWInt)data).m_value );
       else
           return error(data,"/",line);
    }

    public MWDataType mod(MWDataType data, int line) throws MWException
    {
       if( data instanceof MWFloat )
           return new MWFloat( m_value % ((MWFloat)data).m_value );
       if( data instanceof MWInt )
           return new MWFloat( m_value % (float)((MWInt)data).m_value );
       else
           return error(data,"%",line);
    }
}


******************************
MWInt.java
******************************

package internal;

/**
 * Mapwad Int type
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWInt.java,v 1.21 2003/12/18 19:32:36 avrum Exp $
 */

public class MWInt extends MWDataType
{
    int m_value;

    public MWInt()
    {  }

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

    public MWInt(int value)
    {
       m_value = value;
    }

    public boolean passByValue()
    { return true; }

    public int getValue()
    { return m_value; }

    void setValue(int value)
    { m_value=value; }

    public String typeName()
    { return "int"; }

    public String toString()
    { return Integer.toString(m_value); }

    public boolean isCompatibleWith(MWDataType type)
    {
```

```
    if(type instanceof MWInt || type instanceof MWFloat)
        return true;

    return false;
}

public MWDataType assign(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        m_value = ((MWInt)data).m_value;
        return this;
    }
    else if( data instanceof MWFloat )
    {
        m_value = (int)((MWFloat)data).m_value;
        return this;
    }

    return error(data,"=",line);
}

public MWDataType equals(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value == ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value == ((MWFloat)data).m_value);
    }

    return error(data,"==",line);
}

public MWDataType lessThan(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value < ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value < ((MWFloat)data).m_value);
    }

    return error(data,"<",line);
}

public MWDataType greaterThan(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value > ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value > ((MWFloat)data).m_value);
    }

    return error(data,">",line);
}

public MWDataType lessThanEq(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value <= ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value <= ((MWFloat)data).m_value);
    }

    return error(data,"<=",line);
}

public MWDataType greaterThanEq(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        return new MWBoolean(m_value >= ((MWInt)data).m_value);
    }
    else if( data instanceof MWFloat )
    {
        return new MWBoolean(m_value >= ((MWFloat)data).m_value);
    }

    return error(data,">=",line);
}

public MWDataType plusEq(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        m_value += ((MWInt)data).m_value ;
        return this;
    }
    else if( data instanceof MWFloat )
    {
        m_value += ((MWFloat)data).m_value;
        return this;
    }
    else
        return error(data,"+=",line);
}

public MWDataType minusEq(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        m_value -= ((MWInt)data).m_value ;
        return this;
    }
    else if( data instanceof MWFloat )
    {
        m_value -= ((MWFloat)data).m_value;
        return this;
    }
    else
        return error(data,"-=",line);
}

public MWDataType timesEq(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        m_value *= ((MWInt)data).m_value ;
        return this;
    }
    else if( data instanceof MWFloat )
    {
        m_value *= ((MWFloat)data).m_value;
        return this;
    }
    else
        return error(data,"*=",line);
}

public MWDataType divideEq(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        m_value /= ((MWInt)data).m_value ;
        return this;
    }
    else if( data instanceof MWFloat )
    {
        m_value /= ((MWFloat)data).m_value;
        return this;
    }
    else
        return error(data,"/=",line);
}

public MWDataType modEq(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
    {
        m_value %= ((MWInt)data).m_value ;
        return this;
    }
    else if( data instanceof MWFloat )
    {
        m_value %= ((MWFloat)data).m_value;
        return this;
    }
    else
        return error(data,"%=",line);
}

public MWDataType plusPlus(int line)
{
    m_value++;
    return this;
}

public MWDataType minusMinus(int line)
{
    m_value--;
    return this;
}

public MWDataType plusSign(int line)
{
    return new MWInt(+m_value);
}

public MWDataType minusSign(int line)
{
    return new MWInt(-m_value);
}

public MWDataType add(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
        return new MWInt( m_value + ((MWInt)data).m_value );
    else if( data instanceof MWFloat )
        return new MWFloat( (float)m_value + ((MWFloat)data).m_value);
    else
        return error(data,"+",line);
}

public MWDataType subtract(MWDataType data,int line) throws MWException
{
    if( data instanceof MWInt )
        return new MWInt( m_value - ((MWInt)data).m_value );
    else if( data instanceof MWFloat )
        return new MWFloat( (float)m_value - ((MWFloat)data).m_value);
    else
        return error(data,"-",line);
}
```

```
    public MWDataType multiply(MWDataType data,int line) throws MWException
    {
       if( data instanceof MWInt )
           return new MWInt( m_value * ((MWInt)data).m_value );
       else if( data instanceof MWFloat )
           return new MWFloat( (float)m_value * ((MWFloat)data).m_value);
       else
           return error(data,"/",line);
    }

    public MWDataType divide(MWDataType data,int line) throws MWException
    {
       if( data instanceof MWInt )
           return new MWInt( m_value / ((MWInt)data).m_value );
       else if( data instanceof MWFloat )
           return new MWFloat( (float)m_value / ((MWFloat)data).m_value);
       else
           return error(data,"/",line);
    }

    public MWDataType mod(MWDataType data,int line) throws MWException
    {
       if( data instanceof MWInt )
           return new MWInt( m_value % ((MWInt)data).m_value );
       else if( data instanceof MWFloat )
           return new MWFloat( (float)m_value % ((MWFloat)data).m_value);
       else
           return error(data,"%",line);
    }

}


******************************
MWString.java
******************************

package internal;

/**
 * Mapwad string type
 *
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWString.java,v 1.10 2003/12/18 19:32:38 avrum Exp $
 */

public class MWString extends MWDataType
{
    String m_str;

    public MWString()
    {
       m_str="";
    }

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

    public void setValue(String name)
    { m_str=name; }

    public String getValue()
    { return m_str; }

    public String toString()
    { return m_str; }
    public String typeName()
    { return "string"; }

    public boolean isCompatibleWith(MWDataType type)
    {
        if(type instanceof MWString)
            return true;

        return false;
    }

    public MWDataType assign(MWDataType data, int line) throws MWException
    {
        if( data instanceof MWString )
        {
            m_str = ((MWString)data).m_str;
            return this;
        }

        return error(data,"=",line);
    }
}
```

8.1.11. Internal Processing

```
******************************
MWException.java
******************************

package internal;

/**
 * Mapwad compiler exception class
 * Anytime one of these is thrown, we get a compile error
 *
 * @author Ben Smith (bhs16@cs.columbia.edu)
 * @version $Id: MWException.java,v 1.7 2003/12/18 20:02:44 avrum Exp $
 */

public class MWException extends Exception
{
    int m_line;

    public MWException(String msg)
    {
        super(msg);
    }

    public MWException(int line, String msg)
    {
        super(Integer.toString(line)+": "+msg);
        m_line=line;
    }

    public int getLine()
    {
        return m_line;
    }
}


******************************
MWGui.java
******************************

package internal;

import java.lang.Math;
import java.util.*;
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.event.*;

/**
 * Gui that translates a list of walls into lines and points
 *
 * Given a list of walls, where each wall has an associated length and
 * angle to the next wall, this GUI will display its graphical representation.
 *
 * @author Josh Weinberg
 * @version CVS $Id: MWGui.java,v 1.20 2003/12/18 19:32:36 avrum Exp $
 */

public class MWGui
{
    public MWMap map;

    // window size
    static final int MAIN_WIDTH = 600;
    static final int MAIN_HEIGHT = 600;

    // map size
    static final int MAP_WIDTH = 450;
    static final int MAP_HEIGHT = 450;

    // GUI objects
    public JScrollPane scroller, botScroller;
    public static JPanel mainPanel;
    public JPanel panel, sidePanel, legend, bottomPanel, warnPanel;
    public JLabel label;
    public JButton zoom, zoomFit;
    public JTextField zoomAmt;

    /**
     * Constructor for the MWGui object.
     *
     * This constructor will create the canvas with the graphical
     * representation of a list of walls.
     * @param map The map to display.
     */
    public MWGui(MWMap map)
    {
       this.map = map;
       drawMap();
    }

    /**
     * Creates the main panel that will be added to the Frame.
     *
     * This panel will have 3 subpanels:
     * 1) Map window, which is scrollable
     * 2) Control window which includes a legend and zoom controls
     * 3) A console for displaying runtime warnings.
     *
     * @return The main panel that is to be displayed.
     */
    private JPanel makePanel()
    {
       panel = new JPanel();
       panel.setPreferredSize(new Dimension(MAIN_WIDTH, MAIN_HEIGHT));


       /* Panel running down the right side of the GUI */

       sidePanel = new JPanel(new GridBagLayout());
       GridBagConstraints spc = new GridBagConstraints();

       spc.gridx = 0;
       spc.gridy = 0;
       spc.insets = new Insets(0,0,0,5);
       label = new JLabel("Zoom:");
       sidePanel.add(label, spc);

       spc.gridx = 1;
       spc.gridy = 0;
       spc.insets = new Insets(0,0,0,0);
       zoomAmt = new JTextField(3);
       sidePanel.add(zoomAmt, spc);

       spc.gridx = 2;
       spc.gridy = 0;
       spc.insets = new Insets(0,0,0,0);
       label = new JLabel("%");
       sidePanel.add(label, spc);

       spc.gridx = 0;
       spc.gridy = 1;
       spc.gridwidth = 3;
       spc.insets = new Insets(10,0,0,0);
       zoom = new JButton("Zoom");
       sidePanel.add(zoom, spc);

       spc.gridx = 0;
       spc.gridy = 2;
       spc.gridwidth = 3;
       spc.insets = new Insets(10,0,50,0);
       zoomFit = new JButton("Zoom Fit");
       sidePanel.add(zoomFit, spc);

       /* Panel for the legend */

       legend = new JPanel(new GridBagLayout());
       GridBagConstraints lc = new GridBagConstraints();
       legend.setBorder((TitledBorder)BorderFactory.createTitledBorder("Legend"));

       lc.gridx = 0;
       lc.gridy = 0;
       lc.anchor = GridBagConstraints.LINE_START;
       lc.insets = new Insets(10,5,0,5);
       label = new JLabel("X - Thing");
       label.setForeground(Color.red);
       legend.add(label, lc);

       lc.gridx = 0;
       lc.gridy = 1;
       lc.anchor = GridBagConstraints.LINE_START;
       lc.insets = new Insets(5,5,10,5);
       label = new JLabel("S - MapStart");
       label.setForeground(Color.green);
       legend.add(label, lc);

       spc.gridx = 0;
       spc.gridy = 3;
       spc.gridwidth = 3;
       spc.insets = new Insets(50,0,0,0);
       sidePanel.add(legend, spc);

       // create the bottom panel
       makeBottomPanel();

       /* Scroll Pane and button listeners*/
       Mapper map = new Mapper();

       scroller = new JScrollPane(map);
       scroller.setPreferredSize(new Dimension(MAP_WIDTH, MAP_HEIGHT));
       scroller.setMinimumSize(new Dimension(MAP_WIDTH, MAP_HEIGHT));

       // add listeners for the buttons
       zoom.addActionListener(map);
       zoomFit.addActionListener(map);
```

```
        panel.setLayout(new GridBagLayout());
        GridBagConstraints pc = new GridBagConstraints();

        pc.gridx = 0;
        pc.gridy = 0;
        pc.anchor = GridBagConstraints.FIRST_LINE_START;
        pc.insets = new Insets(20,20,10,10);
        scroller.revalidate();
        panel.add(scroller, pc);
        panel.revalidate();

        pc.gridx = 1;
        pc.gridy = 0;
        pc.anchor = GridBagConstraints.LINE_END;
        pc.insets = new Insets(20,10,10,20);
        panel.add(sidePanel, pc);
        panel.revalidate();

        pc.gridx = 0;
        pc.gridy = 1;
        pc.gridwidth = 2;
        pc.anchor = GridBagConstraints.PAGE_END;
        pc.insets = new Insets(10,20,20,20);
        botScroller.revalidate();
        panel.add(bottomPanel, pc);
        panel.revalidate();

        return panel;
    }

    /**
     * Creates the scrollable console window.
     */
    private void makeBottomPanel()
    {
        bottomPanel = new JPanel(new GridBagLayout());
        GridBagConstraints bpc = new GridBagConstraints();

        warnPanel = new JPanel();
        warnPanel.setBackground(Color.white);
        warnPanel.setLayout(new BoxLayout(warnPanel,BoxLayout.PAGE_AXIS));

        label = new JLabel("RUN-TIME CONSOLE",JLabel.CENTER);
        label.setForeground(Color.black);

        botScroller = new JScrollPane(warnPanel);
        botScroller.setPreferredSize(new Dimension(MAIN_WIDTH-25, 75));
        botScroller.setMinimumSize(new Dimension(MAIN_WIDTH-25, 75));

        bpc.gridx = 0;
        bpc.gridy = 0;
        bpc.anchor = GridBagConstraints.PAGE_START;
        bpc.insets = new Insets(0,0,5,0);
        bottomPanel.add(label,bpc);

        bpc.gridx = 0;
        bpc.gridy = 1;
        bpc.anchor = GridBagConstraints.PAGE_END;
        bottomPanel.add(botScroller,bpc);
    }

    /**
     * Creates the frame and its contents and displays them to user.
     */
    public void drawMap()
    {
        try
        {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.gtk.GTKLookAndFeel");
        }
        catch(Exception exc)
        {
        }
        mainPanel = makePanel();
        JFrame frame = new JFrame();
        WindowListener l = new WindowAdapter()
            {
               public void windowClosing(WindowEvent e)
               {
                   System.exit(0);
               }
            };

        frame.addWindowListener(l);
        frame.setContentPane(mainPanel);
        frame.setTitle("Mapper");
        frame.pack();
        frame.setVisible(true);
        frame.repaint();
        frame.setLocationRelativeTo(null);
    }

    /**
     * Inner class that handles the drawing of the Map and includes
     * button pressing event handling.
     */
    class Mapper extends JPanel implements ActionListener
    {
        private MWWalls walls;
        private Vector things;
        private MWLocation mapStart;
        private Vector warnings;

        // values for point size
        private int diameter = 16;

        // values for canvas size and offsets
        private float scale = 1f;
        private float fitScale = 0;
        private int sizeX = 0;
        private int sizeY = 0;
        private int padX = 50, padY = 50;
        private double minX = 0, minY = 0;
        private double maxX = 0, maxY = 0;

        // event values
        boolean fitToWindow = false;

        /**
         * Constructor for the Mapper object.
         */
        public Mapper()
        {
            this.walls = map.getRoom().getWalls();
            this.things = map.getThings();
            this.mapStart = map.getMapStart();
            this.warnings = map.getWarnings();

            // add the warnings to the gui
            processWarnings();

            // find max and min x,y for canvas size
            for(int i=0;i<walls.getNumWalls(); i++)
            {
               Point2D.Float startCoord = walls.getWall(i).getStartCoord();

               double currX = startCoord.getX();
               double currY = startCoord.getY();

               if(currX < minX)
               {
                   minX = currX;
               }
               if(currY < minY)
               {
                   minY = currY;
               }
               if(currX > maxX)
               {
                   maxX = currX;
               }
               if(currY > maxY)
               {
                   maxY = currY;
               }
            }

            fitScale = (float)Math.min((MWGui.MAP_WIDTH-padX)/(maxX-minX),
                                      (MWGui.MAP_HEIGHT-padY)/(maxY-minY));
        }

        /**
         * Method that deals with events
         *
         * @param event The event that occured.
         */
        public void actionPerformed(ActionEvent event)
        {
            Object source = event.getSource();

            // check which button fired the event
            if(source == zoom)
            {
               float zoomFactor = 0;

               try
               {
                   fitToWindow = false;
                   zoomFactor = Float.parseFloat(zoomAmt.getText());
                   scale = (float)(zoomFactor/100);
                   repaint();
               }
               catch(NumberFormatException exc)
               {
               }
               zoomAmt.setText(""+(100*scale));
               return;
            }

            if(source == zoomFit)
            {
               fitToWindow = true;
               repaint();
               return;
            }
        }

        /**
         * Adds the warnings to the bottom panel.
         */
        private void processWarnings()
        {
            MWWarning temp;
            for(int i=0; i<warnings.size(); i++)
            {
               temp = (MWWarning)warnings.get(i);
               label = new JLabel(temp.getWarning());
               label.setForeground(temp.getColor());
               warnPanel.add(label);
            }
        }
```

```
/**
 * Standard paint method for this canvas.
 *
 * This is the method run when the canvas is to be drawn.
 *
 * @param g The Graphics objects that JVM sends to canvas for drawing.
 */
protected void paintComponent(Graphics g)
{
    Graphics2D g2 = (Graphics2D)g;
    super.paintComponent(g);
    setBackground(Color.white);

    double x=0,y=0,x1=0,y1=0;

    int numWalls = walls.getNumWalls();

    if(numWalls < 3)
    {
       return;
    }

    if(fitToWindow)
    {
       scale = fitScale;
    }

    zoomAmt.setText("" + (100*scale));

    // get the preferred size of the panel
    sizeX = (int)Math.ceil((maxX-minX)*scale+padX);
    sizeY = (int)Math.ceil((maxY-minY)*scale+padY);

    // scale the diameter and radius of the points
    int diam = (int)Math.ceil(diameter * scale);
    int radius = (int)Math.ceil(diam / 2);

    // compute offsets for x and y direction
    int offX = 0, offY = 0;
    if(minX < 0)
    {
       offX = (int)Math.ceil(padX/2) + (int)(Math.abs(minX) * scale);
    }
    else
    {
       offX = (int)Math.ceil(padX/2) - (int)(minX * scale);
    }

    if(minY < 0)
    {
       offY = (int)Math.ceil(padY/2) + (int)(Math.abs(minY) * scale);
    }
    else
    {
       offY = (int)Math.ceil(padY/2) - (int)(minY * scale);
    }

    // draw the points and lines to the canvas
    for(int i=0; i<numWalls; i++)
    {
       MWWall temp = walls.getWall(i);
       Point2D.Float startCoord = temp.getStartCoord();
       Point2D.Float endCoord = temp.getEndCoord();

       x = (startCoord.getX() * scale) + offX;
       y = sizeY - ((startCoord.getY() * scale) + offY);
       x1 = (endCoord.getX() * scale) + offX;
       y1 = sizeY - ((endCoord.getY() * scale) + offY);

       // get the color of this wall
       if(temp.getOrigRoom().getColor() != null)
```

```
                {
                    g2.setColor(temp.getOrigRoom().getColor());
                }
                else
                {
                    g2.setColor(Color.blue);
                }

                g2.draw(new Line2D.Double((x+radius),
                                         (y+radius),
                                         (x1+radius),
                                         (y1+radius)));
                g2.draw(new Ellipse2D.Double(x,
                                            y,
                                            diam,
                                            diam));
                g2.draw(new Ellipse2D.Double(x1,
                                            y1,
                                            diam,
                                            diam));
            }

            // print out markers for the things
            if(things != null)
            {
                g2.setColor(Color.red);
                // draw an X where a thing will be placed
                for(int i=0; i<things.size(); i++)
                {
                    Point2D.Float temp =
((MWThing)things.get(i)).getPosition().getCoord();
                    if(temp != null)
                    {
                        x = (temp.getX() * scale) + offX;
                        y = sizeY - ((temp.getY() * scale) + offY);
                        g2.drawString("X", (int)x, (int)y);
                    }
                }
            }

            // print out marker for the start coordinate
            if(mapStart != null)
            {
                g2.setColor(Color.green);
                Point2D.Float temp = mapStart.getCoord();

                if(temp != null)
                {
                    x = (temp.getX() * scale)+ offX;
                    y = sizeY - ((temp.getY() * scale) + offY);
                    g2.drawString("S", (int)x, (int)y);
                }
            }

            // set the preferred size of the window and revalidate the panels
            setPreferredSize(new Dimension(sizeX, sizeY));
            scroller.revalidate();
            mainPanel.revalidate();
        }
    }
}


******************************
MWMap.java
******************************

package internal;

import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.geom.*;

/**
 * Overall Map data structure as created in
 * Map() function.  In charge of actually outputting
 * Quake .map file based on contents of MWMap
 *
 * @author Avrum Tilman, Josh Weinberg, Ron Weiss
 * @version $Id: MWMap.java,v 1.33 2003/12/18 20:00:37 avrum Exp $
 */

public class MWMap
{
    private MWLocation MapStart;
    private float MapHeight = 10;

    private String CeilingTexture = "GROUND1_5";
    private String FloorTexture   = "GROUND1_5";
    static final int SCALE = 10;

    private LinkedList Walls;
    private Vector Things;
    private Vector warnings;

    private ColorGenerator colorGenerator;

    private MWRoom Room;


    //Quake Map parameters
    private static final String gfxwad = "quake101.wad";


    /**
     * MWMap constructor
     *
     * @param rList list of Rooms in the map
     * @param things list of Things in the map
     * @param startloc player starting location within the map
     */
    public MWMap(Vector rList, Vector things, MWLocation startloc)
       throws MWException
    {
       colorGenerator = new ColorGenerator();
       warnings = new Vector();

       MWRoom r = gen(rList, things, startloc);

       Room = r;

       Walls = r.getWalls().getWalls();
       MapStart = startloc;
       Things = things;
    }


    /**
     * MWMap constructor
     *
     * @param rList list of Rooms in the map
     * @param things list of Things in the map
     * @param startloc player starting location within the map
     * @param height height of the map
     */
    public MWMap(Vector rList, Vector things, MWLocation startloc, MWFloat height)
       throws MWException
    {
       colorGenerator = new ColorGenerator();
       warnings = new Vector();
    MWRoom r = gen(rList, things, startloc);

    Room = r;

    Walls = r.getWalls().getWalls();
    MapStart = startloc;
    if(height != null && height.getValue() != 0)
        MapHeight = height.getValue();
    Things = things;
}

/**
 * MWMap constructor
 *
 * @param rList list of Rooms in the map
 * @param things list of Things in the map
 * @param startloc player starting location within the map
 * @param height height of the map
 * @param cTexture texture to use for the map ceiling
 * @param fTexture texture to use for the map floor
 */
public MWMap(Vector rList, Vector things, MWLocation startloc, MWFloat height,
            MWString cTexture, MWString fTexture)
    throws MWException
    {
    colorGenerator = new ColorGenerator();
    warnings = new Vector();

    MWRoom r = gen(rList, things, startloc);

    Room = r;

    Walls = r.getWalls().getWalls();
    MapStart = startloc;
    Things = things;

    if(height != null && height.getValue() != 0)
        MapHeight = height.getValue();

    if(cTexture != null && cTexture.getValue() != null)
        CeilingTexture = cTexture.getValue();

    if(fTexture != null && fTexture.getValue() != null)
            FloorTexture = fTexture.getValue();
}


/**
 * Get method for Things member variable
 *
 * @return List of the Things contained in this MWMap
 */
public Vector getThings()
{
    return Things;
}

/**
 * Get method for Walls member variable
 *
 * @return List of the Walls comprising this MWMap
 */
public LinkedList getWalls()
{
    return Walls;
}

/**
 * Get method for Room member variable
 *
 * @return Final Room comprising the entire map
```

```
     */
    public MWRoom getRoom()
    {
       return(Room);
    }

    /**
     * Get method for MapStart member variable
     *
     * @return Player start location in the map
     */
    public MWLocation getMapStart()
    {
       return(MapStart);
    }

    /**
     * Get method for the warnings generated during processing.
     *
     * @return The list of warnings.
     */
    public Vector getWarnings()
    {
       return warnings;
    }

    /**
     * Method called when there is one master room left after processing,
     * that performs any lingering attaches between walls.
     *
     * If walls that are "near" each other are to be attached, the walls are
     * removed (ie they are attached). If they are not near each other or for
     * some reason if they are not part of the same room, the program is
     * terminated
     *
     * @param r The tom to clean up.
     * @throws MWException If the walls are not in the same room or are not
     * near each other.
     */
    private static void cleanUpWalls(MWRoom r) throws MWException
    {
       MWWalls walls = r.getWalls();

       for(int i=0;i<walls.getNumWalls();i++)
       {
           MWWall check=walls.getWall(i);
           if (check.getIsEntry().getValue())
           {
               if (!check.getConnectedTo().getFromRoom().equals(check.getFromRoom()))
               {
                   throw new MWException("Connected Walls are not in the same room");
               }

               if (check.isNear(check.getConnectedTo()))
               {
                   walls.remove(check);
                   i--; //don't miss a wall
               }
               else
               {
                   throw new MWException("Connected Walls can not be placed near each
other");
               }
           }
       }
    }

    /**
     * Generates a complete map from a list of walls and things.
     *
     * The generation algorithm connects each room one by one (ie removing two
     * room from the list and then inserting 1) until there is only one large
     * room left (ie the main map). It also adds things and a start location
     * to the map.
     *
     * @param rList The list of rooms that will be used to generate the map.
     * @param tList The list of things that are to be placed in the map.
     * @param mapStart The start location in the map.
     * @return MWRoom containing the final map.
     * @throws MWException If MapStart is connected to a wall will be removed
     * because it is an entry wall.
     * @throws MWException If a Room is found to have less than 3 walls.
     */
    private MWRoom gen(Vector rList, Vector tList, MWLocation mapStart) throws
MWException
    {
        if (mapStart.getNearWall().getIsEntry().getValue())
        {
            throw new MWException("MapStart can not be based on an entry wall");
        }

        int size=rList.size();
        for(int i=0;i<size;i++)
        {
            MWRoom temp = (MWRoom)rList.get(i);

            // terminate the program is a room has less than 3 walls
            if(temp.getWalls().getNumWalls() < 3)
            {
                throw new MWException("A Room was found with less than 3 walls");
            }

            if(!temp.isClosed().getValue())
            {
                //warning
                Color color = colorGenerator.getColor();
                temp.setColor(color);
                warnings.add(new MWWarning("WARNING: Room is not closed! Will be closed
automatically.", color));
                temp.close();
            }

            temp.fixWalls();
            temp.setCoords();
        }


        while(rList.size()>1)
        {
            // remove a room and check for an entry wall within
            MWRoom rm1 = (MWRoom)rList.firstElement();
            rList.remove(rm1);
            int w=rm1.getEntryWall();

            // if an entry wall is found try and merge the two connected rooms
            if(w>=0)
            {
                MWWall w1=rm1.getWalls().getWall(w);
                MWWall w2=w1.getConnectedTo();
                MWRoom rm2=w2.getFromRoom();


                w1.align(true);
                w2.align(false);

                w1.merge();

                rList.remove(rm2);
                rList.add(rm1);
            }
            else
            {
                if (rm1.equals(mapStart.getNearWall().getFromRoom()))
                {
                    // check to see if there are any unconnected rooms
                    if(rList.size()>0)
                    {
                        warnings.add(new MWWarning("WARNING: " + rList.size() + "
unconnected room(s) found! They will be discarded."));
                    }

                    // check to see if the final map is singular
                    if(MWTransform.isSelfIntersected(rm1.getWalls()))
                    {
                        warnings.add(new MWWarning("WARNING: Map is self-intersected! There
are overlaping walls."));
                    }

                    // create coordinates for each Thing in the map
                    if(tList != null)
                    {
                        for(int x = 0; x < tList.size(); x++)
                        {
                            MWThing t = (MWThing)tList.get(x);
                            if(t.getPosition().getNearWall().getFromRoom() != rm1 ||
                               getLocationCoords(t.getPosition()).equals(""))
                            {
                                warnings.add(new MWWarning("WARNING: Thing
("+t.getType()+") location is not contained in the map! Thing will be discarded."));
                                tList.remove(x);
                                x--;
                            }
                        }

                    }
                    cleanUpWalls(rm1);
                    return rm1;
                }
                else
                {
                    // check to see if there are any unconnected rooms
                    warnings.add(new MWWarning("WARNING: Unconnected room found! It will
be discarded."));
                }
            }
        }
        if(rList.size()==1)
        {
            MWRoom room = (MWRoom)rList.firstElement();

            if(room.equals(mapStart.getNearWall().getFromRoom()))
            {
                // check to see if the final map is singular
                if(MWTransform.isSelfIntersected(room.getWalls()))
                {
                    warnings.add(new MWWarning("WARNING: Map is self-intersected! There
are overlaping walls."));
                }

                //create coordinates for each Thing in the map
                if(tList != null)
                {
                    for(int x = 0; x < tList.size(); x++)
                    {
                        MWThing t = (MWThing)tList.get(x);
                        if(t.getPosition().getNearWall().getFromRoom() != room ||
                           getLocationCoords(t.getPosition()).equals(""))
                        {
                            warnings.add(new MWWarning("WARNING: Thing (" + t.getType() +
") location is not contained in the map! Thing will be discarded."));
                            tList.remove(x);
                            x--;
                        }
                    }
                }
                cleanUpWalls(room);
                return room;
            }
        }
        return null;
    }


    /**
     * Creates Quake MAP file based on the contents of this map
     *
     * @param filename The name of the output file.
     * @throws MWException If the MapStart has an invalid location (ie not
     * contained inside the map).
     * @throws IOException If there is an error writing the output file.
     */
    public void createMapFile(String filename) throws IOException, MWException
    {
        FileWriter f = new FileWriter(filename);
        Brush[] brushes = new Brush[Walls.size()];
        BrushPoint maxPoint = new BrushPoint(0,0,0);
        BrushPoint minPoint = new BrushPoint(0,0,0);

        // foreach wall convert wall to brush
        for(int x = 0; x < brushes.length; x++)
            brushes[x] = new Brush((MWWall)Walls.get(x), (int)MapHeight*SCALE);

        // output header information
        f.write("{\n  \"sounds\" \"1\"\n  \"classname\" \"worldspawn\"\n  \"wad\" \""
                +gfxwad+"\"\n  \"worldtype\" \"0\"\n");

        // foreach brush, output brush.
        for(int x = 0; x < brushes.length; x++)
        {
            f.write(brushes[x].toString());

            // find max x and max y coordinate to make floor and ceiling that
            // cover entire map
            if(brushes[x].endPoint.x > maxPoint.x)
                maxPoint.x = brushes[x].endPoint.x;
            if(brushes[x].endPoint.y > maxPoint.y)
                maxPoint.y = brushes[x].endPoint.y;
            if(brushes[x].endPoint.x < minPoint.x)
                minPoint.x = brushes[x].endPoint.x;
            if(brushes[x].endPoint.y < minPoint.y)
                minPoint.y = brushes[x].endPoint.y;
        }


        //make floor and ceiling - one huge brush that encloses the entire map
        minPoint.z = (int)MapHeight*SCALE;
        maxPoint.z = (int)MapHeight*SCALE+5;
        Brush ceiling = new Brush(minPoint, maxPoint, CeilingTexture);

        minPoint.z = -5;
        maxPoint.z = 0;
        Brush floor = new Brush(minPoint, maxPoint, FloorTexture);

        f.write(ceiling.toString()+floor.toString());

        f.write("}\n\n");


        // foreach Thing convert to thingbrush
        if(Things != null)
        {
            for(int x = 0; x < Things.size(); x++)
            {
                MWThing t = (MWThing)Things.get(x);
```

```
                String tName = t.getType().getValue();
                String loc = t.getPosition().toString();

                if(!loc.equals(""))
                    f.write("\n{\n  \"classname\" \""+tName+"\"\n  \"origin\"
"+loc+"\n}\n");
            }
        }


        // output player start location
        String startloc = getLocationCoords(MapStart);

        if(startloc.equals(""))
        {
            throw new MWException("MapStart is invalid! Not contained inside map");
        }

        f.write("{\n  \"classname\" \"info_player_start\"\n  \"origin\" "
                +startloc+"\n}");

        f.close();
    }

   /**
    * Finds coordinates of a given MWLocation with respect to its "NearWall".
    *
    * The coordinates will be based on the coordinates of its NearWall, it's
    * position along the wall, and it's offset from the wall.
    *
    * @param l Find x/y coordinates of l (based on Wall/wallpos/offset)
    * @return A string containing the coordinates to use in .map output file
    */
    private String getLocationCoords(MWLocation l)
    {

        // default coords
        double xi = 50;
        double yi = 50;
        double xo = 50;
        double yo = 50;

        if(l != null)
        {
            if(!l.getNearWall().getIsEntry().getValue())
            {
                MWWall w = l.getNearWall();
                double offset = l.getWallDistance().getValue()*SCALE;

                // angle from wall to location of thing with respect to x axis
                double normAngle = 0;

                double deltax = w.getEndCoord().getX() - w.getStartCoord().getX();
                double deltay = w.getEndCoord().getY() - w.getStartCoord().getY();;

                if(deltax <= 0 && deltay >= 0)
                    normAngle = 90 - Math.toDegrees(Math.atan(-deltay/deltax));
                // 90 < normAngle < 180
                else if(deltax > 0 && deltay > 0)
                    normAngle = Math.toDegrees(Math.atan(deltay/deltax)) - 90;
                // 180 < normAngle < 270
                else if(deltax >= 0 && deltay <= 0)
                    normAngle = 270 -  Math.toDegrees(Math.atan(-deltay/deltax));
                // normAngle > 270
                else if(deltax < 0 && deltay < 0)
                    normAngle = Math.toDegrees(Math.atan(deltay/deltax)) - 270;


                double cosA = Math.cos(Math.toRadians(normAngle));
                double sinA = Math.sin(Math.toRadians(normAngle));
                double basex = 0;
                double basey = 0;

                if(l.getWallPosition().getValue().equals("start"))
                {
                    basex = w.getStartCoord().getX();
                    basey = w.getStartCoord().getY();
                }
                else if(l.getWallPosition().getValue().equals("end"))
                {
                    basex = w.getEndCoord().getX();
                    basey = w.getEndCoord().getY();
                }
                else if(l.getWallPosition().getValue().equals("center"))
                {
                    if(deltax >= 0)
                       basex = w.getStartCoord().getX() + Math.abs(deltax)/2;
                    else
                       basex = w.getStartCoord().getX() - Math.abs(deltax)/2;

                    if(deltay >=0)
                       basey = w.getStartCoord().getY() + Math.abs(deltay)/2;
                    else
                       basey = w.getStartCoord().getY() - Math.abs(deltay)/2;
                }


                // get possible coordinates
                if(normAngle <= 90)
                {
                    xi = basex - cosA * offset;
                    yi = basey - sinA * offset;

                    xo = basex + cosA * offset;
                    yo = basey + sinA * offset;
                }
                else if(normAngle > 90 && normAngle <= 180)
                {
                    xi = basex - cosA * offset;
                    yi = basey + sinA * offset;

                    xo = basex + cosA * offset;
                    yo = basey - sinA * offset;
                }
                else if(normAngle > 180 && normAngle <= 270)
                {
                    xi = basex + cosA * offset;
                    yi = basey + sinA * offset;

                    xo = basex - cosA * offset;
                    yo = basey - sinA * offset;
                }
                else if(normAngle > 270)
                {
                    xi = basex + cosA * offset;
                    yi = basey - sinA * offset;

                    xo = basex - cosA * offset;
                    yo = basey + sinA * offset;
                }


                // create polygon out of a room to ensure that location
                // is inside the room
                MWWall currWall, prevWall;

                MWWalls walls = w.getFromRoom().getWalls();

                GeneralPath gp = new GeneralPath();
                gp = MWTransform.convertToPath(walls);
```

```
            if(gp.contains(xi, yi))
                l.setCoord((float)xi, (float)yi);
            else if(gp.contains(xo, yo))
                l.setCoord((float)xo, (float)yo);
            else
            {
                // not contained in room (ie. distance from wall places
                // it outside of room - warning - cannot use location
                return("");
            }

            return(l.toString());
        }
    }
    return("");
}


private class Brush
{
    private BrushPlane top;
    private BrushPlane bottom;
    private BrushPlane inside;
    private BrushPlane outside;
    private BrushPlane left;
    private BrushPlane right;

    private BrushPoint endPoint = null; //origin;

    private String texture;

    private static final int WALL_THICKNESS = 1;


    /**
     * Converts a Mapwad Wall to a Map brush -
     * works fine for walls with angle between 55 and 155 degrees between
     * them - bugs otherwise...
     *
     * @param w Wall to convert
     * @param height height of Wall <- can just store this in each Brush?
     */
    public Brush(MWWall w, int height)
    {
        double length = w.getLength().getValue()*SCALE;
        BrushPoint o, e, p1, p2, p3, p4;
        double cosA, sinA, cosB, sinB;//, tanT;
        double x1, y1, x2, y2, z;

        o  = new BrushPoint(w.getStartCoord());
        p3 = new BrushPoint(w.getEndCoord());

        double deltax = p3.x - o.x;
        double deltay = p3.y - o.y;

        //cumulative angle relative the x axis
        double normAngle = 0;

        //first check special cases (0 x or y offset)
        if (Math.round(deltax) == 0 && deltay > 0)
        {
            normAngle = 90;
        }
        else if (Math.round(deltax) == 0 && deltay < 0)
        {
            normAngle = 270;
        }
        else if (Math.round(deltay) == 0 && deltax > 0)
        {
            normAngle = 180;
        }
```

```
else if (Math.round(deltay) == 0 && deltax < 0)
{
   normAngle = 0;
}
 //normAngle < 90
else if(deltax < 0 && deltay > 0)
{
   normAngle = Math.toDegrees(Math.atan(-deltay/deltax));
}
//90 < normAngle < 180
else if(deltax > 0 && deltay > 0)
{
   normAngle = 180 - Math.toDegrees(Math.atan(deltay/deltax));
}
//180 < normAngle < 270
else if(deltax > 0 && deltay < 0)
{
   normAngle = 180 + Math.toDegrees(Math.atan(-deltay/deltax));
}
//normAngle > 270
else if(deltax < 0 && deltay < 0)
{
   normAngle = 360 - Math.toDegrees(Math.atan(deltay/deltax));
}


//need to take into account direction of new wall for other points
//=> need to split into 4 cases depending on normAngle
//opposite point in rectangular solid - top, outside, end
//depends on normAngle:
if(normAngle <= 90)
{
   sinB = Math.sin(Math.toRadians(90 - normAngle));
   cosB = Math.cos(Math.toRadians(90 - normAngle));

   p2 = new BrushPoint(o.x + cosB * WALL_THICKNESS,
                       o.y + sinB * WALL_THICKNESS, 0);
   e = new BrushPoint(p3.x + cosB * WALL_THICKNESS,
                      p3.y + sinB * WALL_THICKNESS,
                      height);
}
else if(normAngle > 90 && normAngle <= 180)
{
   sinB = Math.sin(Math.toRadians(normAngle - 90));
   cosB = Math.cos(Math.toRadians(normAngle - 90));

   p2 = new BrushPoint(o.x + cosB * WALL_THICKNESS,
                       o.y - sinB * WALL_THICKNESS, 0);
   e = new BrushPoint(p3.x + cosB * WALL_THICKNESS,
                      p3.y - sinB * WALL_THICKNESS,
                      height);
}
else if(normAngle > 180 && normAngle <= 270)
{
   sinB = Math.sin(Math.toRadians(270 - normAngle));
   cosB = Math.cos(Math.toRadians(270 - normAngle));

   p2 = new BrushPoint(o.x - cosB * WALL_THICKNESS,
                       o.y - sinB * WALL_THICKNESS, 0);
   e = new BrushPoint(p3.x - cosB * WALL_THICKNESS,
                      p3.y - sinB * WALL_THICKNESS,
                      height);
}
else //normAngle > 270
{
   sinB = Math.sin(Math.toRadians(normAngle - 270));
   cosB = Math.cos(Math.toRadians(normAngle - 270));

   p2 = new BrushPoint(o.x - cosB * WALL_THICKNESS,
                       o.y + sinB * WALL_THICKNESS, 0);
   e = new BrushPoint(p3.x - cosB * WALL_THICKNESS,
                          p3.y + sinB * WALL_THICKNESS,
                          height);
    }

    p1 = new BrushPoint(o.x, o.y, 0);
    p4 = new BrushPoint(e.x, e.y, 0);


    x1 = p1.x;
    y1 = p1.y;
    x2 = p3.x;
    y2 = p3.y;
    left = new BrushPlane(new BrushPoint(x1, y1, 0),
                         new BrushPoint(x2, y2, 0),
                         new BrushPoint(x1, y1, 1));
    x1 = p2.x;
    y1 = p2.y;
    x2 = p4.x;
    y2 = p4.y;
    right = new BrushPlane(new BrushPoint(x1, y1, 0),
                          new BrushPoint(x1, y1, 1),
                          new BrushPoint(x2, y2, 0));
    x1 = p1.x;
    y1 = p1.y;
    x2 = p2.x;
    y2 = p2.y;
    inside = new BrushPlane(new BrushPoint(x1, y1, 0),
                           new BrushPoint(x1, y1, 1),
                           new BrushPoint(x2, y2, 0));
    x1 = p3.x;
    y1 = p3.y;
    x2 = p4.x;
    y2 = p4.y;
    outside = new BrushPlane(new BrushPoint(x1, y1, 0),
                            new BrushPoint(x2, y2, 0),
                            new BrushPoint(x1, y1, 1));

    //these two never change because we're only transforming x and y
    bottom  = new BrushPlane(new BrushPoint(0, 0, o.z),
                            new BrushPoint(1, 0, o.z),
                            new BrushPoint(0, 1, o.z));
    top     = new BrushPlane(new BrushPoint(0, 0, e.z),
                            new BrushPoint(0, 1, e.z),
                            new BrushPoint(1, 0, e.z));

    texture = w.getTexture().getValue();

    endPoint    = e;  //but we need the inside at height 0...
    endPoint.z  = 0;
}


/**
 * Brush constructor.  Creates a brush based on 2 points -
 * opposite diagonal corners of a rectangular solid region
 * this isnt gonna be rotated...
 *
 * @param origin
 * @param end
 * @param t name of texture to use on this brush
 */
public Brush(BrushPoint origin, BrushPoint end, String t)
{
    texture = t;

    left    = new BrushPlane(new BrushPoint(origin.x, 0, 0),
                            new BrushPoint(origin.x, 1, 0),
                            new BrushPoint(origin.x, 0, 1));
    right   = new BrushPlane(new BrushPoint(end.x, 0, 0),
                            new BrushPoint(end.x, 0, 1),
                            new BrushPoint(end.x, 1, 0));
```

```
        inside  = new BrushPlane(new BrushPoint(0, origin.y, 0),
                                 new BrushPoint(0, origin.y, 1),
                                 new BrushPoint(1, origin.y, 0));
        outside = new BrushPlane(new BrushPoint(0, end.y, 0),
                                 new BrushPoint(1, end.y, 0),
                                 new BrushPoint(0, end.y, 1));
        bottom  = new BrushPlane(new BrushPoint(0, 0, origin.z),
                                 new BrushPoint(1, 0, origin.z),
                                 new BrushPoint(0, 1, origin.z));
        top     = new BrushPlane(new BrushPoint(0, 0, end.z),
                                 new BrushPoint(0, 1, end.z),
                                 new BrushPoint(1, 0, end.z));
    }

    public String toString()
    {
        String brush;

        String indent = "    ";

        if(texture == "")
           texture = "bricka2_6";

        brush = "  {\n";
        //              plane                   texture  texture offset/scale
        brush += indent+left.toString()   +" "+texture+" 0 0 0 1.0 1.0\n";
        brush += indent+right.toString()  +" "+texture+" 0 0 0 1.0 1.0\n";
        brush += indent+inside.toString() +" "+texture+" 0 0 0 1.0 1.0\n";
        brush += indent+outside.toString()+" "+texture+" 0 0 0 1.0 1.0\n";
        brush += indent+bottom.toString() +" "+texture+" 0 0 0 1.0 1.0\n";
        brush += indent+top.toString()    +" "+texture+" 0 0 0 1.0 1.0\n";
        brush += "  }\n";

        return(brush);
    }
}

private class BrushPlane
{
    private BrushPoint  p1,p2,p3;

    public BrushPlane(BrushPoint a, BrushPoint b, BrushPoint c)
    {
        //ensure 3 points aren't colinear -
         //not necessary - guaranteed by Brush contructors above

        p1 = a;
        p2 = b;
        p3 = c;
    }

    public String toString()
    {
        return(p1.toString()+" "+p2.toString()+" "+p3.toString());
    }
}

private class BrushPoint
{
    private double x,y,z;

    public BrushPoint(int xi, int yi, int zi)
    {
        x = xi;
        y = yi;
        z = zi;
    }

    public BrushPoint(double xi, double yi, double zi)
    {
        x = xi;
        y = yi;
        z = zi;
    }

    public BrushPoint(java.awt.geom.Point2D.Float p)
    {
        x = (double)p.getX();
        y = (double)p.getY();
        z = 0;
    }

    public String toString()
    {
        //always round up:
        return("( "+Math.round(x)+" "+Math.round(y)+" "+Math.round(z)+" )");
    }

    public Object clone()
    {
        return((Object)(new BrushPoint(x,y,z)));
    }
}

/**
 * Class for generating distinct colors.
 *
 * @author Josh Weinberg
 */
class ColorGenerator
{
    private Vector red;
    private Vector green;
    private Vector blue;

    public ColorGenerator()
    {
        red = new Vector();
        green = new Vector();
        blue = new Vector();
    }

    public Color getColor()
    {
        int newRed, newGreen, newBlue;

        a:while(true)
        {
            newRed = (int)(256*Math.random());

            for(int i=0; i<red.size(); i++)
            {
                if(newRed == ((Integer)red.get(i)).intValue())
                {
                    continue a;
                }
            }

            red.add(new Integer(newRed));
            break;
        }

        g:while(true)
        {
            newGreen = (int)(256*Math.random());

            for(int i=0; i<green.size(); i++)
            {
                if(newGreen == ((Integer)green.get(i)).intValue())
                {
                    continue g;
                }
                }

                green.add(new Integer(newGreen));
                break;
            }

            b:while(true)
            {
                newBlue = (int)(256*Math.random());

                for(int i=0; i<blue.size(); i++)
                {
                    if(newBlue == ((Integer)blue.get(i)).intValue())
                    {
                        continue b;
                    }
                }

                blue.add(new Integer(newBlue));
                break;
            }

            return new Color(newRed, newBlue, newGreen);
        }
    }
}


******************************
MWTransform.java
******************************

package internal;

import java.awt.*;
import java.awt.geom.*;

/**
 * Class that deals with transformations of vectors in space.
 *
 * These transformations include translations and rotations.
 * @author Josh Weinberg
 * @version CVS $Id: MWTransform.java,v 1.7 2003/12/18 19:32:39 avrum Exp $
 */

public class MWTransform
{
    /**
     * Translates every wall in walls by the given offsets.
     *
     * The offsets given will be subtracted from each point in the
     * list of walls.
     * @param walls The list of walls to translate.
     * @param offX The offset in the x-direction to translate.
     * @param offY The offset in the y-direction to translate.
     */
    public static void translate(MWWalls walls, float offX, float offY)
    {
        int numWalls = walls.getNumWalls();

        Point2D.Float startCoord;
        Point2D.Float endCoord;

        for(int i=0; i< numWalls; i++)
        {
            MWWall temp = walls.getWall(i);

            // get current coordinates
            startCoord = temp.getStartCoord();
            endCoord = temp.getEndCoord();
            // reset coordinates
            temp.setStartCoord((float)startCoord.getX()-offX,
                               (float)startCoord.getY()-offY);
            temp.setEndCoord((float)endCoord.getX()-offX,
                             (float)endCoord.getY()-offY);
        }
    }

    /**
     * Rotates every wall in walls around z-axis by the given angle.
     *
     * Looking down onto the x-y plane, a rotation of a positive angle
     * is a counter-clockwise rotation of the walls around the center.
     * The rotate method works by left multiplying 2x2 rotation matrix with
     * a 1x2 position vector. The rotation vector is of the following form:
     * cos(@)  -sin(@)
     * sin(@)  cos(@)
     * where @ is the angle of rotation.
     * @param walls The list of walls to rotate.
     * @param angle The angle of rotation.
     */
    public static void rotate(MWWalls walls, float angle)
    {
        // create rotation matrix
        float topX = (float)Math.cos(Math.toRadians(angle));
        float topY = -1 * (float)Math.sin(Math.toRadians(angle));
        float botX = (float)Math.sin(Math.toRadians(angle));
        float botY = (float)Math.cos(Math.toRadians(angle));

        int numWalls = walls.getNumWalls();
        for(int i=0; i<numWalls; i++)
        {
            MWWall temp = walls.getWall(i);

            // get coordinates
            float startX = (float)(temp.getStartCoord()).getX();
            float startY = (float)(temp.getStartCoord()).getY();
            float endX = (float)(temp.getEndCoord()).getX();
            float endY = (float)(temp.getEndCoord()).getY();

            // apply rotation
            float newStartX = (startX * topX) + (startY * topY);
            float newStartY = (startX * botX) + (startY * botY);
            float newEndX = (endX * topX) + (endY * topY);
            float newEndY = (endX * botX) + (endY * botY);

            // reset coordinates
            temp.setStartCoord(newStartX, newStartY);
            temp.setEndCoord(newEndX, newEndY);
        }
    }

    /**
     * Computes the angle of rotation that the wall should be rotated.
     *
     * This rotation angle will make it so the endpoint lies along the
     * postive y-axis. As explained in rotate method, rotation angle
     * is counter-clockwise when veiwing the x-y plane from above.
     * @param endPointX The current position of the endpoint x-coord.
     * @param endPointY The current position of the endpoint y-coord.
     * @return The angle of rotation in degrees.
     */
    public static float getRotationAngle(float endPointX, float endPointY)
    {
        float angle = 0;
        float hyp = (float)Point2D.Float.distance(0,0,endPointX,endPointY);

        if(endPointY > 0)
        {
            if(endPointX > 0) // end point in first quadrant
            {
                angle = (float)Math.toDegrees(Math.acos(endPointY/hyp));
            }
            else // end point in second quadrant
            {
                angle = (float)Math.toDegrees(Math.acos(endPointY/hyp)) * -1;
            }
        }
        else
        {
            endPointY *= -1;
            if(endPointX > 0) // end point in fourth quadrant
            {
                angle = 180 - (float)Math.toDegrees(Math.acos(endPointY/hyp));
            }
            else // end point in third quadrant
            {
                angle = 180 + (float)Math.toDegrees(Math.acos(endPointY/hyp));
            }
        }
        return angle;
    }

    /**
     * Reflects a list of walls around the x-axis.
     *
     * @param walls The list of walls to reflect.
     */
    public static void reflectX(MWWalls walls)
    {
        int numWalls = walls.getNumWalls();

        Point2D.Float startCoord;
        Point2D.Float endCoord;

        for(int i=0; i< numWalls; i++)
        {
            MWWall temp = walls.getWall(i);

            // get current coordinates
            startCoord = temp.getStartCoord();
            endCoord = temp.getEndCoord();

            // reset coordinates
            temp.setStartCoord((float)startCoord.getX(),
                               (float)startCoord.getY() * -1);
            temp.setEndCoord((float)endCoord.getX(),
                             (float)endCoord.getY() * -1);
        }
    }

    /**
     * Reflects a list of walls around the y-axis.
     *
     * @param walls The list of walls to reflect.
     */
    public static void reflectY(MWWalls walls)
    {
        int numWalls = walls.getNumWalls();

        Point2D.Float startCoord;
        Point2D.Float endCoord;

        for(int i=0; i< numWalls; i++)
        {
            MWWall temp = walls.getWall(i);

            // get current coordinates
            startCoord = temp.getStartCoord();
            endCoord = temp.getEndCoord();

            // reset coordinates
            temp.setStartCoord((float)startCoord.getX() * -1,
                               (float)startCoord.getY());
            temp.setEndCoord((float)endCoord.getX() * -1,
                             (float)endCoord.getY());
        }
    }

    /**
     * Converts the given list of walls into a closed General Path.
     *
     * @param walls The list of walls to be converted.
     * @return The General Path representation of this room.
     */
    public static GeneralPath convertToPath(MWWalls walls)
    {
        GeneralPath path = new GeneralPath();
        MWWall currWall;

        float err = .001f;

        int numWalls = walls.getNumWalls();

        // get starting point
        MWWall temp = walls.getWall(0);
        path.moveTo((float)(temp.getStartCoord()).getX(),
                    (float)(temp.getStartCoord()).getY());

        for(int i=0; i<numWalls; i++)
        {
            temp = walls.getWall(i);
            path.lineTo((float)(temp.getEndCoord()).getX(),
                        (float)(temp.getEndCoord()).getY());
        }
        path.closePath();

        return path;
    }

    /**
     * Checks if any of the walls in the specified list of walls
     * overlap (ie map has self-intersection.
     *
     * @param walls The list of walls to check.
     * @return Whether or not there is a overlap between walls.
     */
    public static boolean isSelfIntersected(MWWalls walls)
    {
        int numWalls = walls.getNumWalls();

        Line2D.Float line1, line2;
        MWWall wall1, wall2;

        for(int i=0; i<numWalls; i++)
        {
            wall1 = walls.getWall(i);
            line1 = new Line2D.Float(wall1.getStartCoord(), wall1.getEndCoord());
            for(int j=i+1; j<numWalls; j++)
            {
                wall2 = walls.getWall(j);
                line2 = new Line2D.Float(wall2.getStartCoord(), wall2.getEndCoord());

                if(!connected(wall1.getStartCoord(), wall2.getEndCoord()) &&
                   !connected(wall1.getEndCoord(), wall2.getStartCoord()) &&
                   !connected(wall1.getStartCoord(), wall2.getStartCoord()) &&
                   !connected(wall1.getEndCoord(), wall2.getEndCoord()) &&
                   line1.intersectsLine(line2))
                {
                    return true;
                }
            }
        }
```

```
        return false;
    }

    /**
     * Checks to see if the specified points are connected.
     *
     * "Connected" means that the distance between them is less
     * then a specific error margin.
     *
     * @param point1 The first point.
     * @param point2 The seconds point.
     * @return Whether or not the 2 points are connected.
     */
    private static boolean connected(Point2D.Float point1,
                                Point2D.Float point2)
    {
        float err = .001f;
        if((float)point1.distance(point2) < err)
        {
            return true;
        }
        return false;
    }
}


******************************
MWWarning.java
******************************

package internal;

import java.awt.*;

/**
 * Class for keeping track of warnings produced during the processing
 * of the Rooms.
 *
 * @author Josh Weinberg
 * @version CVS $Id: MWWarning.java,v 1.2 2003/12/18 19:32:39 avrum Exp $
 */

class MWWarning
{
    private String warning;
    private Color color;

    private Color defaultColor = Color.black;

    /**
     * Constructor for the MWWarning object.
     *
     * This constructor creates a new warning with the specified string.
     *
     * @param warning The contents of the warning.
     */
    MWWarning(String warning)
    {
        this.warning = warning;
        color = defaultColor;
    }

    /**
     * Constructor for the MWWarning object.
     *
     * This constructor creates a new warning with the specified string,
     * and the specified color.
     *
     * @param warning The contents of the warning.
     * @param color The color associated with the warning.
     */
    MWWarning(String warning, Color color)
    {
       this.warning = warning;
       this.color = color;
    }

    /**
     * Returns the string associated with this warning.
     *
     * @return The string associated with this warning.
     */
    String getWarning()
    {
       return warning;
    }

    /**
     * Returns the color associated with this warning.
     *
     * @return The color associated with this warning.
     */
    Color getColor()
    {
       return color;
    }
}
```

8.2. Mapwad Code Samples

8.2.1. bigmaze.mapwad

```
//Some global variables
int ROWS = 30;
int COLS = 30;
int WIDTH = 10;
int BADGUYS = 100;
int WEAPONS = 20;
int POWERUPS = 50;

string[] textures = {"BRICKA2_4", "BRICKA2_2", "BRICKA2_6", "WBRICK1_5",
                    "TECH04_3", "azwall1_5", "azwall3_1", "azwall3_2",
                    "city1_4", "city2_1", "city2_2", "city2_3", "city2_5",
                    "elwall1_1", "elwall2_4","metal1_5", "metal1_6",
                    "rock3_2", "rock3_7","stone1_5", "stone1_7" };
string[] floors = {"afloor1_3", "afloor1_4", "afloor1_8", "afloor3_1",
                  "GROUND1_1", "GROUND1_2", "GROUND1_5", "GROUND1_8",
                  "sfloor1_2", "sfloor3_2", "sfloor4_1", "sfloor4_8"};
string[] baddies =  {"monster_army", "monster_ogre", "monster_zombie",
                    "monster_dog", "monster_knight", "monster_wizard",
                    "monster_ogre", "monster_demon1", "monster_shambler"};
string[] powerups = {"item_cells", "item_rockets", "item_shells", "item_spikes",
                     "item_health", "item_artifact_super_damage",
                    "item_artifact_invulnerability",
                    "item_artifact_invisibility", "item_armorInv",
                    "item_armor2", "item_armor1"};
string[] weapons =  {"weapon_supershotgun", "weapon_nailgun",
                    "weapon_supernailgun", "weapon_grenadelauncher",
                    "weapon_rocketlauncher", "weapon_lightning"};
string[] misc =     {"misc_fireball", "misc_explobox"};


/*
 *  FourWall()
 *  Creates a 4 wall regular room
 */
Room FourWall(float length, string texture="BRICKA2_2")
{
    Room newRoom = Room();

    int i;
    for(i=0;i < 4;i++)
    {
       Wall wall = Wall(length, 90, "", texture);

       if(i == 0)
       {
           wall.Name = "bottom";
       }
       else if(i == 1)
       {
           wall.Name = "right";
       }
       else if(i == 2)
       {
           wall.Name = "top";
       }
       else if(i == 3)
       {
           wall.Name = "left";
       }

       Add(newRoom,wall);
    }

    Close(newRoom);

    return newRoom;
}

//creates some random things in random locations
randThings(Room[][] rooms, string[] thingTypes, int numThings)
{
    int num = 1+RandInt(numThings);
    int c;
    for(c = 0; c < num; c++)
    {
       int x = RandInt(ROWS);
       int y = RandInt(COLS);
       int wall = RandInt(4);

       Thing(thingTypes[RandInt(thingTypes.Size)],
             Location(rooms[x][y].Walls[wall], "center", WIDTH/2));
    }

}

/*
 * Generates a random maze
 */
Map()
{
    Room[][] rooms = Room[ROWS][COLS];

    int x;
    int y;
    for(x = 0; x < COLS; x++)
    {
       for(y = 0; y < ROWS; y++)
       {
           rooms[x][y] = FourWall(WIDTH, textures[RandInt(textures.Size)]);
       }
    }

    for(x=0;x < COLS-1;x++)
    {
       for(y=0;y < ROWS-1;y++)
       {
           int direction = RandInt(2);
```

```
            int t = RandInt(3);

            if(direction == 1 && x == COLS-1)
            {
               direction = 0;
            }
            else if(direction == 0 && y == ROWS-1)
            {
               direction = 1;
            }

            if(direction == 0) //attach top
            {
               Attach(rooms[x][y].Walls["top"],
                          rooms[x][y+1].Walls["bottom"], "center");
            }
            else if(direction == 1) //attach right
            {
               Attach(rooms[x][y].Walls["right"],
                      rooms[x+1][y].Walls["left"], "center");
            }
        }
    }

    Wall startwall = rooms[0][0].Walls[0];

    randThings(rooms, baddies, BADGUYS);
    randThings(rooms, weapons, WEAPONS);
    randThings(rooms, powerups, POWERUPS);

    //Set the location where the player enters the map
    MapStart = Location(startwall, "center", WIDTH/2);
}
```

8.2.2. hallways.mapwad

```
/* A trapeziodal room that when combined with 7 other identical trapezoids
 * forms a octagon shaped hallway
 */
Room trap(float inner, float width, string texture="GROUND1_2")
{
    float y=width/(Tan(67.5)); //calculate the extra size of the bottom wall
    float z=Sqrt(width*width+y*y); //calculate the size of the side wall
    Room ret = Room();
    Add(ret,Wall(inner+2*y,67.5,"out",texture));
    Add(ret,Wall(z,112.5,"right",texture));
    Add(ret,Wall(inner,112.5,"inner",texture));
    Add(ret,Wall(z,67.5,"left",texture));

    return ret;
}

/* Creates an octagon shaped hallway by combing 8 trapezoids
 */
Room[] pentagon(float inner, float width, string texture="GROUND1_2")
{
    Room[] rooms = Room[8];
    int i;

    for(i=0;i<8;i++)
    {
       rooms[i]=trap(inner,width,texture); //Add the room to the array
       if (i>0)
       {
           //Attach the room to its neighbor
           Attach(rooms[i-1].Walls["right"],rooms[i].Walls["left"]);
       }
    }
    //close the hallway
    Attach(rooms[7].Walls["right"],rooms[0].Walls["left"]);
    return rooms;
}

//Calculates the outer diameter of the pentagon hallway
float pentOuterDiam(float inner, float width)
{
    float y=width/(Tan(67.5));
    return (2*((inner+2*y)/(Sqrt(2)))+inner+2*y);
}

//Calculates the diameter of the inner courtyard in the hallway
float pentInnerDiam(float inner, float width)
{
    return (2*(inner/Sqrt(2))+inner);
}

/**
 * Given the dimensions of a hallway and the corridor size calculates the
 * needed size of the next hallway
 */
float getNextInner(float inner, float width, float cor)
{
    return ((Sqrt(2)*(2*cor+pentOuterDiam(inner,width)))/(2+Sqrt(2)));
}

//Creates the corridor room
Room corridor(float length, float width, string texture="WBRICK1_5")
{
    Room r = Room();
    Add(r,Wall(length,90, ,texture));
    Add(r,Wall(width,90,"right",texture));
    Add(r,Wall(length,90,, texture));
    Add(r,Wall(width,90,"left",texture));

    return r;
}

Map()
{
    int i;
    int n;
    float width=5;
    float cor=4;
    float corWidth=width;
    float inner=5;

    //All the possible wall textures
    string[] textures = {"BRICKA2_4", "BRICKA2_2", "BRICKA2_6", "WBRICK1_5",
                        "TECH04_3", "azwall1_5", "azwall3_1", "azwall3_2",
                        "city1_4", "city2_1", "city2_2", "city2_3", "city2_5",
                        "elwall1_1", "elwall2_4","metal1_5", "metal1_6",
                        "rock3_2", "rock3_7","stone1_5", "stone1_7" };

    //creates the inner most hallway
    Room[] innerOct=pentagon(inner,width,textures[RandInt(textures.Size)]);
    MapStart=Location(innerOct[7].Walls["out"],"center",2);

    //create 3 more hallways outside of the first one
    for(n=0;n<3;n++)
    {
       inner=getNextInner(inner,width,cor); //get the next inner size
       //create the next octagon
       Room[] outerOct=pentagon(inner,width, textures[RandInt(textures.Size)]);

       Room[] cors=Room[4]; //create the 4 corridor rooms

       for(i=0;i<4;i++)
       {
           cors[i]=corridor(cor,corWidth);
       }
```

```
        //now attach the corridors between the hallways
        for(i=0;i<4;i++)
        {
            //n%2 switches which walls the hallways are on.
            Attach(innerOct[i*2+n%2].Walls["out"],cors[i].Walls["right"]);
            Attach(outerOct[i*2+n%2].Walls["inner"],cors[i].Walls["left"]);
        }
        innerOct=outerOct;
    }

    CeilingTexture = "sky1";
}
```

8.2.3. recursive2.mapwad

```
/*
 * AnyRoom(int walls, float length=0, float[] lengths={0})
 *
 * Returns a room with 'walls' number of walls.  There are two ways to
 * specify wall length.  If 'length' is used, then all walls have their
 * Length variable set to 'length'.  When 'lengths' is used, then the length
 * of each wall is passed in to the function.  'length' and 'lengths'
 * cannot both be used in a single call (obviously).
 * i.e. AnyRoom(5,10) builds a pentagon shaped room with each wall
 * 10 units long
 *
 */
Room AnyRoom(int walls, float length=0,float[] lengths=null)
{
        Room newRoom = Room();
        int i;
        float angle = 180*(walls-2)/walls;
        Wall wall;

        if(lengths == null)
        {
          lengths=float[walls];

          for(i=0;i < walls;i++)
          {
            lengths[i]=length;
          }
        }

        for(i=0;i < walls;i++)
        {
          wall=Wall();

          wall.Length=lengths[i];
          wall.AngleToNext=angle;
          wall.Texture="city2_2";

          Add(newRoom,wall);
        }

        Close(newRoom);

        return newRoom;
}

Map()
{
    recurseGen();

//    CeilingTexture="sky1";
}

recurseGen(Wall wall=null, float size=20)
{
    Print(size);
    if(size <= 2)
    {
       return;
    }

    int i;
    int thingIndex=RandInt(5);

    Room rmFirst;
    Room rmLast;
    Room rmNew;

    float[] lens = {size/2,size,size/2,size};

    for(i=0;i < 6;i++)
    {
       rmNew = AnyRoom(6,size);

       if(i==0)
       {
           rmFirst = rmLast = rmNew;
       }

       Attach(rmLast.Walls[2],rmNew.Walls[0]);

       if(i==thingIndex)
       {
           Thing("monster_zombie",Location(rmNew.Walls[1],,size/4));
       }

       if(i % 2 == 0)
       {
           recurseGen(rmNew.Walls[4],size/2);
       }

       rmLast=rmNew;
    }

    Attach(rmLast.Walls[2],rmFirst.Walls[0]);


    if(wall == null)
    {
       MapStart = Location(rmFirst.Walls[1], "center",  5);
    }
    else
    {
       Room hall = AnyRoom(4,,lens);

       Attach(wall,hall.Walls[0]);
       Attach(hall.Walls[2],rmLast.Walls[4],"center");
    }
}
```

8.2.4. wavyhalls.mapwad

```
Room sinRoom(int width, int segments)
{
    Room sinR=Room();
    Wall one= Wall(width,225-180,"start");

    Add(sinR,one);

    int x=0;

    int y=20;

    for (x=0 ;x<360*segments;x+=18 )
    {
        Wall w= Wall(10,180+(Sin(x)*(2*y)/3.0 ));
        Add(sinR,w);
    }
    Add(sinR,Wall(width,315-180));

    Wall end=Wall(width,225-180,"start");

    Add(sinR,end);
    Add(sinR,Wall(width,180));

    for (x=0 ;x<360*segments;x+=18 )
    {
        Wall w= Wall(10,180+(Sin(x)*(2*y)/3.0 ));
        Add(sinR,w);
    }

    return sinR;
}


Room polygon(int l,int s)
{
    Room p=Room();
    int angle=180*(s-2)/s;
    int q=0;
    for (q=0;q<s;q++)
    {
        Add(p,Wall(l,angle));
    }

        return p;
}

Room sinSpokes(int w, int s,int seg)
{
    Room t=polygon(w,2*s);

    int q=0;
    for (q=0;q<s;q++)
    {
        Room new = sinRoom(w,seg);
        Attach(t.Walls[2*q],new.Walls["start"]);
    }

    return t;
}

Map()
{
    Room s=sinSpokes(32,3,1);
    MapStart = Location(s.Walls[1], "center", 20);
}
```

8.3. Sample Mapwad.java Output

The intermediate .map file generated by the Mapwad compiler is broken down into two main parts. It consists of a list of 'entities' contained in the map. There are two basic types of entities in the .map file. The first is a map layout entity, containing a list of all of the non-interactive objects (essentially walls, the floor and the ceiling) comprising the map. In this entity is a list of 'brushes' each one corresponding to a wall. Each brush is defined by the intersection of six planes along with their associated texture information. After this is a list of entities representing each Thing in the map, followed by a single entity containing the player start location corresponding to the MapStart variable in the Mapwad code. Each of these entities consists of some text describing what kind of entity it is and its coordinates in the map.

Basic .map file layout:

```
{
  Map layout entity

  {
    Wall brush

    left plane
    right plane
    inside plane
    outside plane
    bottom plane
    top plane
  }
  {
    Wall brush
  }
  ...

  {
    Floor brush
  }
  {
    Ceiling brush
  }
}

{
  Thing entity
}
{
  Thing entity
}
...

{
  MapStart entity
}
```

```
Here is a sample .map file:

{
  "sounds" "1"
  "classname" "worldspawn"
  "wad" "quake101.wad"
  "worldtype" "0"
  {
    ( -1500 -1500 0 ) ( -1400 -1500 0 ) ( -1500 -1500 1 ) rock3_7
0 0 0 1.0 1.0
    ( -1500 -1501 0 ) ( -1500 -1501 1 ) ( -1400 -1501 0 ) rock3_7
0 0 0 1.0 1.0
    ( -1500 -1500 0 ) ( -1500 -1500 1 ) ( -1500 -1501 0 ) rock3_7
0 0 0 1.0 1.0
    ( -1400 -1500 0 ) ( -1400 -1501 0 ) ( -1400 -1500 1 ) rock3_7
0 0 0 1.0 1.0
    ( 0 0 0 ) ( 1 0 0 ) ( 0 1 0 ) rock3_7 0 0 0 1.0 1.0
    ( 0 0 100 ) ( 0 1 100 ) ( 1 0 100 ) rock3_7 0 0 0 1.0 1.0
  }
  {
    ( -1400 -1500 0 ) ( -1300 -1500 0 ) ( -1400 -1500 1 )
azwall3_1 0 0 0 1.0 1.0
    ( -1400 -1501 0 ) ( -1400 -1501 1 ) ( -1300 -1501 0 )
azwall3_1 0 0 0 1.0 1.0
    ( -1400 -1500 0 ) ( -1400 -1500 1 ) ( -1400 -1501 0 )
azwall3_1 0 0 0 1.0 1.0
    ( -1300 -1500 0 ) ( -1300 -1501 0 ) ( -1300 -1500 1 )
azwall3_1 0 0 0 1.0 1.0
    ( 0 0 0 ) ( 1 0 0 ) ( 0 1 0 ) azwall3_1 0 0 0 1.0 1.0
    ( 0 0 100 ) ( 0 1 100 ) ( 1 0 100 ) azwall3_1 0 0 0 1.0 1.0
  }
  ...

}

{
  "classname" "weapon_supershotgun"
  "origin" "-1350 -1150 25"
}
{
  "classname" "weapon_supernailgun"
  "origin" "-1350 -1550 25"
}
...

{
  "classname" "info_player_start"
  "origin" "-1950 -2150 25"
}
```

The actual file is quite large. It was generated using bigmaze.mapwad and contains about 400 walls and 50 things giving a total file size of about 190 kilobytes.