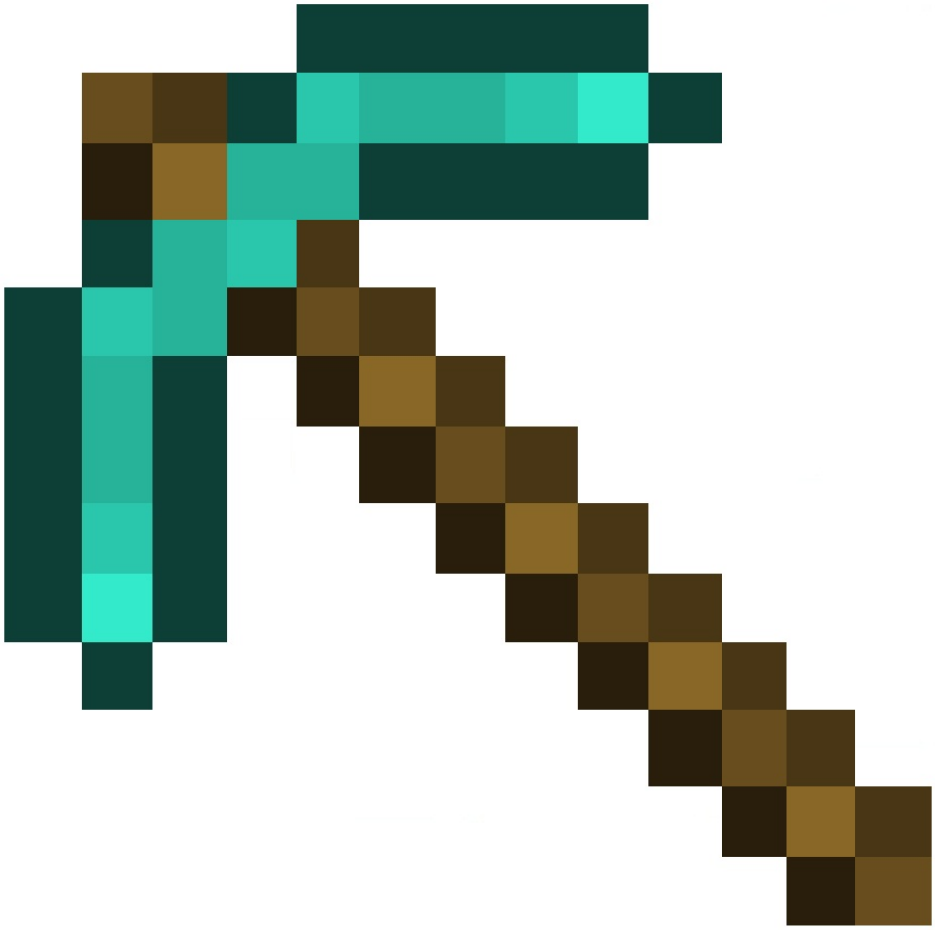
MineTime



Team 18:

Mirza Ali - Project Manager
Tanay Jaipuria - Language and Tools Guru
Don Yu - System Architect
Patrice Liang - System Integrator
Stephen Zhou - System Tester

Table of Contents

White Paper

Minecraft 101

History

Minecraft is a sandbox genre game developed by the independent game company Mojang. Over the past few years, Minecraft has exploded in popularity and has sold over 10 million copies as of April 2013. Content updates are frequent, continually offering players new content.

Gameplay

Minecraft revolves around combat, exploration, harvesting resources, and, most importantly, building structures out of the various types of blocks available in the Minecraft universe.

Building is done by placing collected blocks from the player's inventory onto the Minecraft map. "Vanilla" Minecraft, e.g. Minecraft without using cheats or external editing programs, is typically tedious and time-consuming as each block has to be individually collected and placed. As the scale of a Minecraft project expands, the player has to increasingly worry about proper scaffolding, lighting, and monster attack deterrents.

Figure: A selection of hostile creatures in a fence structure.

Building

There are currently over 170 different blocks that can be used as building material in Minecraft. Each block type has a unique look and feel. Some blocks have other special properties. For example, chests can store other blocks and act like external inventories, torches light up dark areas, water flows radially outward from the source, sand falls down with "gravity," etc.

There are a couple of building restrictions:

1. Blocks can only be placed off of another block. This means blocks cannot be placed on air alone.
2. There is no gravity acting on blocks in Minecraft (except with some special cases like with sand).
3. Players can only place a block within 4 blocks of him or herself.
4. Blocks can only be placed one at a time through pointing and clicking.

Maps

Maps are automatically generated as the player explores an area. As such, maps do not have a defined finite boundary.

All the information associated with a Minecraft map is stored within a directory. The directory is roughly structured as the following:

```
MapName/
    level.dat - global metadata of the map
    players/<player>.dat - information of each player in the map
    data/ - stores map item data
```

External Applications

Because of the potential for complexity in Minecraft structures, there exists programs and libraries that can manipulate directly maps without the typical restrictions found in vanilla Minecraft. In a similar sense, players can use the MineTime programming language to algorithmically manipulate the map.

The Better Way to Craft

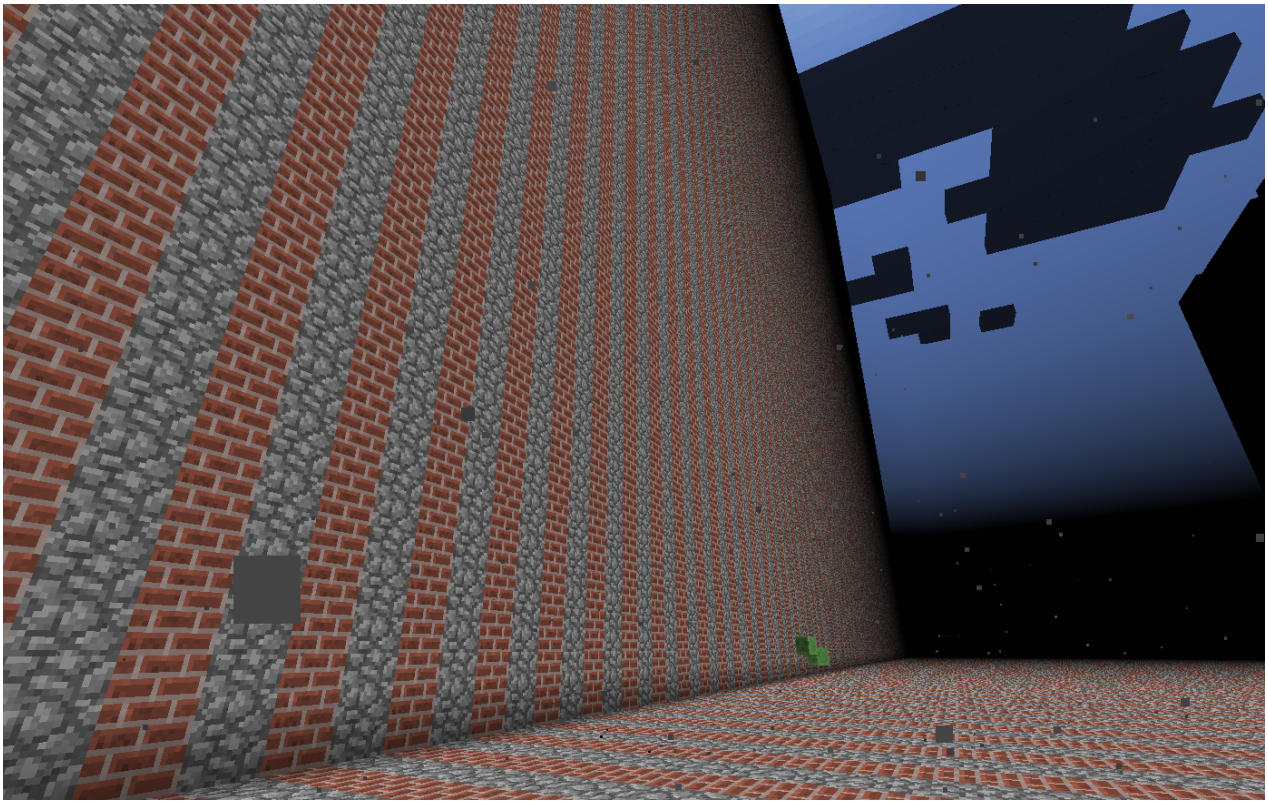


Figure: A cobble and brick wall built with MineTime code.

MineTime simplifies

For example, making a large geometrically simple shape like a circle or rectangle typically requires a fair amount of planning because of both the large scale of the map and the limited reach of the player. A common method for players is to meticulously plan shapes on a gridded template, like on paper or on Excel, in order to ensure symmetry. Using MineTime greatly reduces the planning time. Creating regular structures is as easy as using trigonometric equations within a loop.

MineTime saves time

Players do not have to worry about collecting resources to build structures because MineTime directly modifies the map data when adding blocks.

MineTime is easy

Creating a map is as simple as initializing a FlatMap object. Adding a block is as easy as calling an add method. The syntax and grammar is succinct and simple.

Intended Users

MineTime is intended for both existing and new players of Minecraft. The efficiency of the language appeals to existing users that are already familiar with the game and wish to devote more time to imagining new and complex structures. For new players, MineTime allows them to

quickly dive into the game, as it renders the process of building structures logical and accessible.

In terms of programming background, MineTime is intended for users that possess basic understanding of programming constructs, as the syntax closely mirrors those of Python and Java. Beyond that, MineTime is intuitive and does not require advanced skill sets in programming.

Properties of Language & Buzzwords

A gaming language must be simple and intuitive, so that it will appeal to a majority of users. Below are the design goals of our language:

Elegant

We would like our syntax to be elegant and easy to use. For this purpose, we have tried to keep it similar to Python, which we believe has a very simple and clean syntax.

Simple

In Minecraft, structures are built block-by-block. This makes it very time consuming to build large structures. In our language, one can combine blocks/towers to make a structure before actually needing to place it. Therefore, we have abstracted away the work in figuring out how the coordinates will work to some extent.

Portable

Since the final output of our programming language is a Minecraft map file, any computer that is compatible with Minecraft can enjoy the benefits of the language and can utilize the output of our program. Further, our target language is Python. Therefore any system that can run Python can execute programs written in our language to create map files.

MineTime encapsulates a number of features that make it easy to use and portable, allowing for innovative creation within the game.

Related Languages

There are no programming languages that interact with Minecraft but there are a various applications that do so.

MCTerra

MCTerra is a generator that creates maps of different terrains, including flat maps, biosphere maps, desert maps, golden tunnels maps, and planetoid maps. It aims to speed up the process of generating terrain with user-specified features and settings.

Mace

Mace is a generator that creates a random structures at the spawn point.

MCEdit

MCEdit is an comprehensive GUI application that allows the user to edit and create worlds. Its features include copy, clone, filters, and more.

Why MineTime?

Though the Minecraft community has created many programs and editors to enhance the Minecraft map creation process, MineTime distinguishes itself from all of them with its focus on speed by eliminating repetitive and monotonous actions through function abstractions. Whereas editors like MCTerra and Mace create maps quickly, they don't grant the player the freedom to fully exercise his or her creativity and whim. For example, MCTerra constrains the user to the available settings and features and Mace creates every randomly. Contrast this to Minetime's endless number for possible programs.

MCEdit is the most powerful of the three MineTime program's, but still requires the player to navigate a GUI and drag n' drop structures. MineTime's algorithmic approach to cloning embedded in the language is more efficient for building defined and symmetric structures.

Conclusion

MineTime addresses the specific case of programming from the gamer's perspective rather than from the developer's side. Our language will be built ground-up to enable gamers to reach their goals through intuitive programming logic rather than long unproductive gaming hours. While it would typically require an hour to design and build a 200 block x 200 block wall in Minecraft, a MineTime program, once written and compiled, will do the same in 2 minutes. Furthermore, as the user becomes more familiar with the language, the time required to generate the code would be further reduced.

The better way to develop Minecraft maps is here, now, brought to you by Team 18.

Example

It's simple to build complex Minecraft structures using MineTime's built-ins.

```
// Creates a 500x500 flat Minecraft map
map = FlatMap("testmap", 500, 500);


// Add wall at (0, 0, 0) point
map.add(wall, (0, 0, 0));


// Demonstration using loops


for (int x=0; i < 20; i++) {
      map.add(block(COBBLE), (x, 0, 0));
      for (int y=1; y <=4) {
            map.add(block(BRICK), (x, y, 0));
      }
      map.add(block(STONE), (x, 0, 5);
}
```

Language Tutorial

Introduction

This will be a short tutorial of the MineTime language. Rather than bogging the reader down with details, we will go through example programs that demonstrate the use of MineTime.

This tutorial will focus on the basics of the language - types, constants, expressions, inputs, and outputs, so that the user can get to writing useful programs in MineTime fairly quickly. Experienced programmers should be able to write advanced programs after going through these tutorials. First, we will provide a brief introduction to Minecraft, the game for which MineTime was created.

Minecraft is a sandbox construction game that allows users to create and destroy blocks in a 3D environment. The user explores the virtual world and mines resources to build structures by assembling basic resource blocks, ultimately create their own worlds.

MineTime is a programming language that takes away the drudgery of painstakingly laying down every single Minecraft block so that players can concentrate on what they do best--creating virtual architectural masterpieces. It alleviates the burden of having to rebuild from scratch and enables users to play Minecraft algorithmically as well as extend previous creations, which take the form of MineTime programs. Consider the scenario of building a tower. Originally, a player would have to manually stack blocks repetitively to construct it, but MineTime provides the means to iterate through the placement of blocks using control loops. In addition, recreating previous Minecraft structures is as simple as instantiating an object.

Before we demonstrate a Hello World program, a quick word on the input and output of MineTime.

Input and Output

The purpose of MineTime is to create Minecraft maps more quickly and efficiently. As such, the output will always be a map file, which the user can then open directly in the Minecraft game. The input will be some statements and expressions which are used to construct structures and objects on the map.

For example, a completely minimal program would look like:

```
map = new FlatMap(“testmap”, 500, 500, 500);
map.close();
```

Upon running this program, an empty map would be outputted. Loading this map into Minecraft would be the same as simply starting a new Minecraft game with an empty map which you can build upon in the game.

Now, a simple Hello World program to demonstrate some of the syntax of MineTime.

Hello World

A Hello World program in MineTime creates an empty 500px*500px*500px map and places a cobble block at the origin. This is the simplest demonstration of the language because the primary functionality of MineTime is to output a map, and the placement of the cobble block demonstrates MineTime's capacity to build structures, starting with the most basic block.

Program 1

```
map = FlatMap("testmap", 500, 500, 500); $ Initialise a map of 500px *
500px * 500px
p = new Point(0,0,0);
map.add(block(COBBLESTONE), p); $ Place a block of cobble at the
origin
map.close() $ finish
```

First, let's explain what is going on here.

The first line simply creates an empty map of size 500px * 500px * 500px called "testmap."

The second line creates a point at the origin.

The third line adds one block of cobble at the origin.

A block is the basic data structure of MineTime. It is a 1px*1px*1px block, of a given type. Types include air, stone, brick, cobble and grass.

In general, the add function takes in two parameters - the first is a block and the second is a point where the structure should be added.

The third line simply closes the map, implying that it cannot be further modified.

Another thing you might notice is the "$" string followed by some text. This is the notation used for comments. Comments are ignored by the compiler, and are generally used to make a program easier to understand.

So for example, the "$ Initialize a map of 500px * 500px * 500px" for example, is ignored by the compiler.

Once you are clear with what is going on, save the above program in a file ending with .mt, helloworld.mt for example.

Then, to run the program, use the command

```
./minetime.py helloworld.mt
```

If everything goes as expected, once the program is compiled, upon typing the command ls, you will notice a new file in the directory called “testmap.”

This is the output of the program and can be loaded as a map directly in Minecraft by opening the game, selecting “Single Player,” then selecting “testmap” as the map file to import.

Start the game and load the map if you’d like - but come back to read the rest of the tutorial!

Variables and Expressions

Now we present a program that shows how operators and functions in MineTime work.

```
Program 2
map = new FlatMap(“testmap”, 500, 500, 500);
x = 10;
y = x+5;
z = x+5*2;
b = new Point(x,y,z);
c = block(BRICK);
map.add(c, b);
map.close();
```

This program is similar to the hello world one in that all it does is creates a couple of blocks, but it gives us some more insight into how variables and expressions work.

As is seen above, the language is dynamically typed, and so specifying the type of a variable is not needed. b refers to a point, and x,y and z to numbers.

The first line creates a map, as in the previous program.

The second, third and fourth lines all assign values to variables.

* has higher precedence than +, and so first 5 is multiplied by 2 before being added to x in line 4. z therefore has a value of 20.

The line below, creates a new point with the values in x, y and z and stores it in b.

```
b = new Point(x,y,z);
```

The next line assigns the block type of air to the variable c.

The line below adds the block to the point b.

```
map.add(c, b);
```

And lastly, this line is used to close the map.

```
map.close();
```

At this point, a couple of things should be mentioned.

First, every program should end with the close method being called on the map, or else the blocks added won't be saved to the map.

Control Flow

Loops

Above, we demonstrated how to build a wall using operators and the transpose function. Below we will show how the same wall can be made using loops, and in the process demonstrate how control flows in MineTime work.

The for statement: in the MineTime programming language we use loops to make building constructs easier. We use it below to create a wall.

Creating walls is considered a simple repetition of blocks. First we build in the the z direction and then recreate the construction across the x and y coordinates depending on the user's choice. Let's take a look at a basic example. Note that we are only showing a code snippet and not the entire program so as to focus on the new things here (we have left out the initialization and closing of the map and the main function)

```
$* Demonstration using loops *$

for (x=0; x < 20; x = x+1) {
      t = new Point(x,0,0);
      map.add(block(COBBLE), t);
      for (y=1; y <=4; y=y+1)
      {
            p = new Point(x,y,0);
            map.add(block(BRICK), p);
      }
      top = new Point(x,5,0);
      map.add(block(STONE), top);
}
```

The implementation of block structures is done via nested loops, where the for statement is formatted in a generic way. For the outer loop, we have the initialization of a temporary variable x = 0, a condition x < 20 which limits the size of the given wall, and x=x+1, the increment step after which the condition is re-evaluated. A loop terminates if its condition is false at any time.

The outer loop creates a block of a given type at a particular location and repeats it across the given range for the wall. The inner loop then stacks each of the constructed blocks for a given height. Utilizing nested loops allows the user to create a simple wall in MineTime.

Our language also has while loops.

In a while loop, as long as the condition in the parentheses is true, the statement's in the body are executed.

Here is a code snippet that uses a while loop to build a tower:

```
y = 0;
while (y<20)
{
      p = new Point(0,y,0);
      map.add(block(GLASS), p);
      y = y + 1;
}
```

In the snippet above, y < 20 is true for twenty iterations, and so a tower of height twenty is built since blocks are essentially added one on top of the other twenty times.

If Else Statements

We have seen how loops help the user in coding out a working program. MineTime also supports if else statements so as to properly direct control flow.

```
if (expression)
{
 statements
}
else
{
 statements
}

Below is an example use-case:

if (x>5)
{
      p = new Point(x,0,0);
      map.add(block(STONE), p);
}
else
{
      p = new Point(x,0,0);
```

```
	map.add(block(BRICK), p);
}
```

In the above snippet, if x is greater than 5, a stone block is added. Otherwise, a brick block is added.

The condition in the if has a number of options associated with it. We can check for:

- equals by '=='
- not equals by '!='
- greater than or lesser than by '>' / '<'
- or by '||'

Functions

In MineTime a function is equivalent to a subroutine, and it allows for an easier way to encapsulate a given computation.

Here, we will look at how to define our own functions.

Functions are defined by the following syntax:

def function_name(parameter-list)
{
	function body
}

Here is an example function to create a tower, similar to the tower built in the loops section above.

```
def create_tower(startx, startz, height)
{
	for (i = 0; i < height; i=i+1)
	{
		p = new Point(startx, height, startz);
		map.add(block(GLASS), p);
	}
}
```

This creates a tower of the height specified by looping through and adding glass blocks 'height' number of times.

Now to call this function, we simply use it's name, and give it whatever parameters we want. For example, to create a tower of height 10 at startx = 0, and startz = 0, we would do:

```
create_tower(0,0,10);
```

The nice thing about having written this function is that we can now create walls of different heights in a number of places. The code therefore becomes very reusable, as the example below shows. It creates two different towers in different locations and of different heights.

```
def main()
{
map = FlatMap(“testmap”, 500, 500, 500);
create_tower(0,0,50);
create_tower(10,10,100);
map.close();
}
```

As you might have noticed, this code was written in a main function. The main function is not strictly required, but encouraged, especially when you have multiple functions, some of which you might not be calling. If a main function exists, when the program runs, all the code in it is executed.

A quick note about functions: functions should be declared earlier in the input program than where they are called.

For example, we cannot write:

```
def A(x):
{
      B(x);
}

def B(x):
{
}
```

The compiler would throw an error, since B(x) has not been defined yet. However, the following is fine:

```
def B(x):
{
}

def A(x):
{
      B(x);
}
```

Conclusion

This Language Tutorial covers the core of the MineTime programming language and hopefully will assist users in writing their own programs of varied purposes and sizes. It showcases the essentials of the language and offers a basic understanding of the grammar. A comprehensive analysis can be found in the Language Reference Manual.

Language Reference Manual

Introduction

This manual describes the numerous features that MineTime supplies to make creating complex architecture within Minecraft easier. We start with an overview of the lexical conventions used within the language, follow with the language syntax, and end with a grammar to represent MineTime.

Lexical Conventions

Every program is reduced to a sequence of tokens. Each token can be one of five classes: identifiers, keywords, constants, operators, and separators. In addition, spaces, tabs, newlines, and comments, which we will collectively refer to as whitespace, separate tokens.

Comments

MineTime uses a dollar sign $ for single line comments. Single line comments extend to a newline character. Block comment blocks start with $* and end with *$ and can span multiple lines. A comment block cannot be nested within another comment block.

Identifiers

Identifiers must start with a letter or underscore. Otherwise, it can consist of any sequence of letters, digits, and underscores. Any letters used in the identifier are case-sensitive.

An identifier can represent a primitive, object, function, and class.

Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

if	new
else	true
def	false
for	FlatMap
while	Point
return	Block
import	

Reserved

The following characters are reserved for use in the grammar.

+	(	{	<=
-	)	}	&&

*	=	>	||
/	;	<	!=
[	:	>=	.
]	==	,	$

Types

Basic Types

MineTime has several primitive types.

Integer

Integer constants are the same as the implementation in Python 2.7. int is two's complement, 128 bits in size. long has arbitrary precision, limited only by the memory available on the machine. An int is automatically promoted to a long when its value falls out of the int's bounds.

Float

Floating numbers are the same as the implementation in Python 2.7. float has arbitrary precision and will handle as many decimal places and significant digits as allowed by the available memory on the machine.

Boolean

A boolean encapsulates true and false values.

String

A string is a sequence of characters surrounded by double quotation marks.

Block

The basic building unit in Minecraft is. It is the cubic block of a certain type (i.e. Cobble, Air, Stone, Grass, Dirt). Similarly, we have block constants consisting of a single cubic block that is of constant block type initialized with an integer 1-172. Block constants will always preserve its type.

Derived Types

Lists

A list is an ordered data structure that holds zero or more items. A list can be initialized with a comma-separated list of items surrounded by '[' and ']'. List elements can be accessed using the get() method. Additional items can be added and removed with the append() and delete() methods respectively. Lists are dynamically sized as it grows and shrinks, limited only by the machine's available memory.

Point

A point is a wrapper for a three-integer tuple, representing an x, y, z coordinate in Minecraft. It must contain 3 integers and cannot hold objects of any other type.

Constants

Character Constants

Character constants are a sequence of one or more characters whose value cannot be changed. Character constants are not an actual type; instead, they are implicitly converted to the primitive string.

Certain special constants cannot be represented without conventional keys or are reserved. They can instead be represented with escape sequences.

newline	\n
horizontal tab	\t
carriage return	\r
backslash	\\
single quote	\’
double quote	\”

Boolean Constants

Boolean constants are either true or false.

Scope

Lexical Scope

The lexical scope of an identifier for a declared variable will always start at the end of its declaration and persist until its current scope exits. Variable names in the same scope must be unique.

Global Scope

Variables declared outside of functions exist within all functions and persist through the entirety of the program.

Function Scope

Variables declared inside functions persist within the function and dies upon exiting the function.

Statement Block Scope

Variables declared inside a statement block persist within the block and dies upon exiting the block.

Linkage Scope

Every external library function or constant that is imported to the MineTime program can be accessed through its identifier. Externally linked library identifiers are shared by the entire program that it is imported into.

Expressions

Primary Expressions

Primary expressions are identifiers, constants, strings, or parenthesized expressions.

primary-expression:
ID
STRING
NUMBER
point-gen
'(' *expression* ')'

point-gen:
POINT

Function Calls

A function call consist of the function name with its parameter list surrounded by parentheses.

function-expression:
ID '(' *parameter-list* ')'

Parameters are passed in by value. Argument expressions are evaluated in the order they are specified in the parameter list.

Class References

MineTime supports a couple of built-in class types like FlatMap.

initializer:
'new' ID '(' *parameter-list* ')'
primary-expression

class-method-expression:
ID '.' *function-expression* ';'

An identifier is a class type when initialized with the class name preceded by the keyword new. Class functions are called with the identifier name, period, and function name.

Multiplicative Operators

The multiplicative operators are * and /, and they group left-to-right. The usual type conversions apply for arithmetic operators. Additional cases are noted below.

multiplicative-expression:
primary-expression
multiplicative-expression '*' *primary-expression*
multiplicative-expression '/' primary-expression

Arithmetic operations require operands of arithmetic types.

* represents multiplication. If a string is an operand and an int is another operand, the result is the string repeated the value of the int times.

/ represents division and yields the quotient. If the second operand is 0, an error is thrown. Division with both int and/or long operands results in discarding the fractional portion of the result.

Additive Operators

The additive operators are + and - group left-to-right. The usual type conversions apply for arithmetic operators. Additional cases are noted below.

additive-expression:
multiplicative-expression
additive-expression '+' *multiplicative-expression*
additive-expression '-' *multiplicative expression*

Arithmetic operations require operands of arithmetic types.

+ represents the summation of the left operand and right operand. If the operands are strings, then + represents concatenation.

- represents the left operand subtract the right operand.

Relational Operators

The relationship operators <, >, <=, and >= group left-to-right.

relational-expression:
additive-expression
relational-expression '<' *additive-expression*
relational-expression '>' *additive-expression*
relational-expression '<=' *additive-expression*
relational-expression '>=' *additive-expression*

If the operands are arithmetic, the < (less), > (greater), <= (less or equal), >= (greater or equal) operators return a boolean value true if the specified relation is true and false if it is false.

If both operands are strings, then the same relation operation is carried out with the first string character converted to its ASCII value. For two strings with equal ASCII value, the comparison is carried on with the following character, and so on.

Equality Operators

The equality operators == and != function as the relational operators do, but they have lower precedence. They are used as follows:

equality-expression:

relational-expression
equality-expression '==' *relational-expression*
equality-expression '!=' *relational-expression*

The operands are arithmetic, the == (equal to) and != (not equal to) operators return a boolean value true if the specified relation is true and false if it is false.

If both operands are strings, then the operation is analogous to the relational operators equal with equality and inequality.

If one operand is arithmetic and the other is string, the equality always results in false and inequality always results in true.

Logical AND Operator

The && operator groups left-to-right.

logical-AND-expression:
equality-expression
logical-AND-expression '&&' *equality-expression*

If both operands evaluate to true or a boolean equivalent, the logical OR expression yields a boolean with value true; otherwise, it yields false.

Logical OR Operator

The || operator groups left-to-right.

logical-OR-expression:
logical-AND-expression
logical-OR-expression '||' *logical-AND-expression*

If either of the two operands evaluate to true or a boolean equivalent, the logical OR expression yields a boolean with value true; otherwise, it yields false.

Assignment Expressions

Assignment expressions evaluate right-to-left

assignment-expression:
ID '=' NEW *initializer*
ID '=' *assignment-expression*
logical-OR-expression

The left operand must be an identifier. Since MineTime is dynamically typed, the type of the left operand is the type of the right operand.

Declarations

Declarations specify how an identifier should be interpreted, and do not necessarily reserve storage.

external-declaration:
    function-definition
    statement
function-definition:
    'def' ID '(' *parameter-list* ')' '{' *statement-list* '}'
    'def' ID '(' *parameter-list* ')' '{' '}'

Initialization

Initializers are used to initialize variables to a certain value or to create objects of types such as Point or Map.

initializer:
    ID '(' *parameter-list* ')'
    primary-expression

An object may be initialized upon declaration. The initializer is preceded by the '=' operator and takes the form of an assignment expression.

Statements

Statements do not hold values and are executed for effect. Most statements are expression statements, and these can be nested in a compound statement, which is usually a function definition. If, if-else, for, and while are control flow statements.

statement:
    expression-statement
    compound-statement
    selection-statement
    iteration-statement
    class-method-expression
    return-statement

Expression Statement

The most basic statement.

expression-statement:
    ';'
    expression ';'

Compound Statement

The compound statement allows for function definitions.

compound-statement:
‘{‘ ‘}’
‘{‘ statement-list ‘}’

Selection Statements

Selection statements are a form of control flow.

selection-statement:
‘if’ ‘(‘ *expression* ‘)’ *statement*
‘if’ ‘(‘ *expression* ‘)’ *statement* ‘else’ *statement*

The if statement expression evaluates a boolean or boolean equivalent. If the expression is true, then it executes the first statement.

The if-else statement evaluates the expression and executes the first statement if the expression evaluates to true or the equivalent. Otherwise it executes the second statement. There is an ambiguity with nested if-else. The ambiguity is resolved by binding the else with the nearest if.

Iteration Statements

Iteration statements are a looping technique.

iteration-statement:
‘for’ ‘(‘ *expression-statement expression-statement expression* ‘)’ *statement*
‘while’ ‘(‘ *expression* ‘)’ *statement*

In the for statement, the first expression in the parentheses is evaluated once and serves as an initialization. The second expression is evaluated at the beginning of each iteration of the loop, and if it evaluates to false the loops it terminated. If an expression is omitted for the second expression, then it defaults to true. The third expressions is evaluated at the end of each iteration and serves as a re-initialization and incrementation.

The while statement evaluates the parenthesized expression at the beginning of each iteration and executes the statement if it evaluates to true.

Note that the loops are all equivalent to one another.

Grammar

```
translation-unit:
    external-declaration
    translation-unit external-declaration

external-declaration:
```

```
      function-definition
      statement

function-definition:
      'def' ID '(' parameter-list ')' '{' statement-list '}'
      'def' ID '(' parameter-list ')' '{' '}'

statement:
      expression-statement
      compound-statement
      selection-statement
      iteration-statement
      class-method-expression
      return-statement

compound-statement:
      '{' '}'
      '{' statement-list '}'

statement-list:
      statement
      statement-list statement

expression-statement:
      ';'
      expression ';'

expression:
      assignment-expression;

assignment-expression:
      ID '=' NEW initializer
      ID '=' assignment-expression
      logical-OR-expression

logical-OR-expression:
      logical-AND-expression
      logical-OR-expression '||' logical-AND-expression

logical-AND-expression:
      equality-expression
      logical-AND-expression '&&' equality-expression

equality-expression:
      relational-expression
```

```
      equality-expression '==' relational-expression
      equality-expression '!=' relational-expression

relational-expression:
      additive-expression
      relational-expression '<' additive-expression
      relational-expression '>' additive-expression
      relational-expression '<=' additive-expression
      relational-expression '>=' additive-expression

additive-expression:
      multiplicative-expression
      additive-expression '+' multiplicative-expression
      additive-expression '-' multiplicative expression

multiplicative-expression:
      primary-expression
      multiplicative-expression '*' primary-expression
      multiplicative-expression '/' primary-expression

initializer:
      'new' ID '(' parameter-list ')'
      primary-expression

class-method-expression:
      ID '.' function-expression ';'

parameter-declaration:
      initializer

primary-expression:
      ID
      STRING
      NUMBER
      point-gen
      '[' statement-list ']'
      '(' expression ')'

function-expression:
      ID '(' parameter-list ')'

parameter-list:
      parameter-declaration-optional
      parameter-list ',' parameter-declaration
```

```
point-gen:
      POINT

iteration-statement:
      'for' '(' expression-statement expression-statement
expression ')' statement
      'while' '(' expression ')' statement

selection-statement:
      'if' '(' expression ')' statement
      'if' '(' expression ')' statement 'else' statement

return-statement:
      'return' ';'
      'return' expression ';'
```

Project Plan

Mirza Ali

Our team brainstormed ideas for our programming language during the first week of February. After multiple sessions of debating which idea had the best potential we decided to create MineTime. To help keep track of our development cycle and keep everyone in the loop, we created a GitHub repository and set up a virtual environment for testing on all our systems. This ensured a speedy, up to date and efficient project management system.

Planning

Our team met every Friday to work on our project, recap the material covered in class as well as assign responsibilities and tasks for the upcoming week. After writing the MineTime White paper we used Google Docs to collaborate on deliverables and Asana for task management. This way we were able to plan, organize and stay in sync.

Specifications

Once we had a working understanding of the procedure involved in creating our compiler we met up with our mentor Melanie to show her our objectives and goals for the project plan. After writing up our Language Tutorial and LRM we started working on implementation stage. As per our tasks specification we divided up workloads into creating our grammar, defining the test environment and building our Abstract Syntax Tree.

Development

We followed through the development of MineTime according to the stages outlined in a compiler design. Having set up a fully functional version control system on GitHub, changes were easy and everyone could work independently as well as share with the group. At each stage of the development cycle we created a branch from the master repository so as to have a working implementation. This enabled us to move quickly yet allow us the opportunity to retrace our steps.

Testing

To make sure the entire workflow was smooth and without bugs we unit tested each stage of our implementation. Over a period of time as new features were added, our system tester made sure that everything from start to finish worked as expected. In several scenarios where some commits caused errors we were able to roll back to a previous version and then fix the root of the problem. The testing plan is covered in more detail in chapter 8.

Responsibilities

To collaborate in an efficient and fair manner, everyone’s responsibilities were outlined during the weekly meeting and then updated in our task management system, Asana. Dividing the

workload across the five team members allowed for quick implementation of the stages of our project and also helped us remain organized.

Team Member	Responsibility
· Don Yu	Scope and Type Checking, Semantics
· Mirza Ali	Compiler Front End, Code Generation
· Patrice Liang	Integrating code, Documentation
· Stephen Zhou	Test case creation, Building Grammar
· Tanay Jaipuria	Code generation and AST Traversal

Implementation Style Sheet

While building our project and moving from the design to the implementation phase, we knew that programming environments could vary among the group due to different operating systems. We decided to maintain a programming style consistent with Python as our final code generation was in Python. We used indentation to indicate control structures, tabs to format code, and spaces to keep the code aesthetically pleasing.

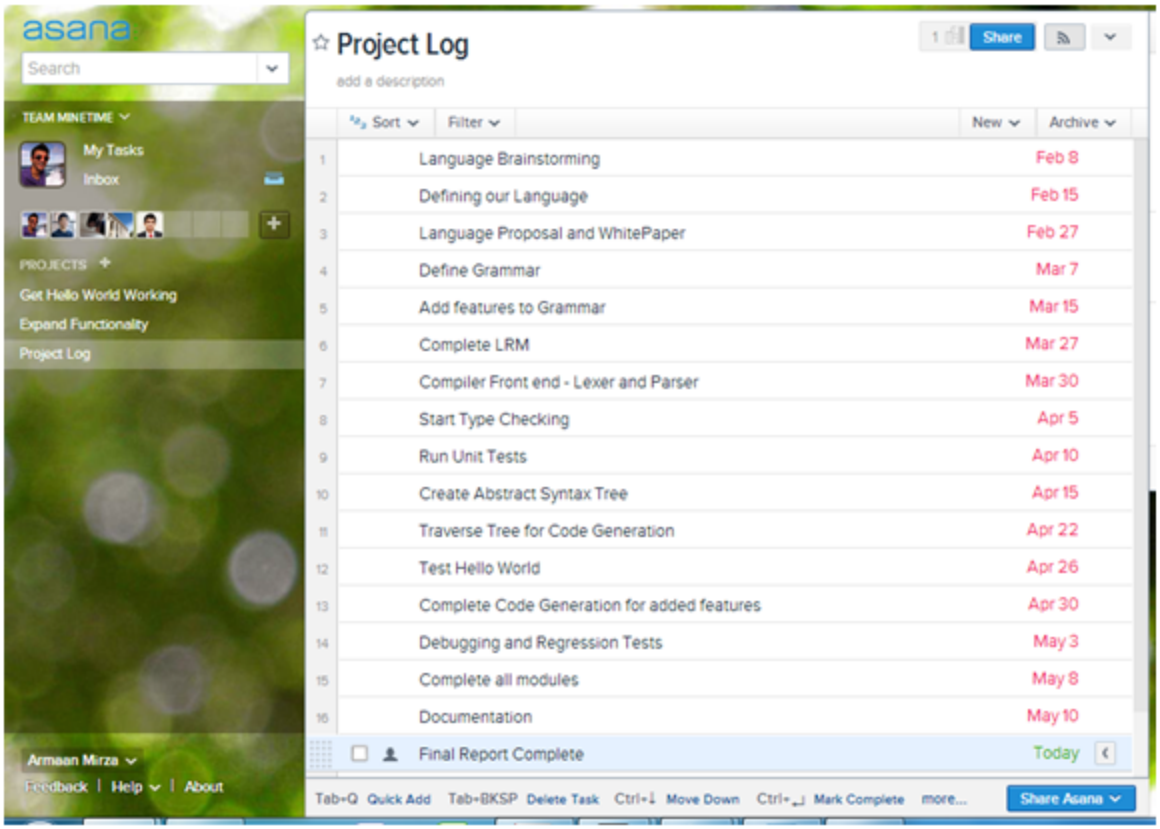
As a group we decided to use a text editor Sublime Text to keep everything consistent across all platforms. We generally commented functions to describe their utility and also used single line comments to explain specific code. This enabled us to keep our code neat and easy to understand by everyone in the team. Another important factor was the use of GitHub to push code and keep commits short and descriptive. This allowed the team to work in a productive and efficient manner.

Timeline

The Project timeline we aimed for is shown in the below table:

Date	Milestone
Feburary 27	Language Proposal and whitepaper Complete
March 27	Language Reference Manual Complete
April 5	Compiler Front End (Lexer and Parser) Complete
April 19	Semantics & Type Checking Complete
April 26	Hello World Test Case Working
May 3	Debugging and Regression Testing
May 12	Final Report Complete

Project Log


asana
Search
TEAM MINETIME
My Tasks
Inbox
PROJECTS
Get Hello World Working
Expand Functionality
Project Log
Armaan Mirza
Feedback | Help | About
Project Log
add a description
Share
Sort
Filter
New
Archive
1 Language Brainstorming Feb 8
2 Defining our Language Feb 15
3 Language Proposal and WhitePaper Feb 27
4 Define Grammar Mar 7
5 Add features to Grammar Mar 15
6 Complete LRM Mar 27
7 Compiler Front end - Lexer and Parser Mar 30
8 Start Type Checking Apr 5
9 Run Unit Tests Apr 10
10 Create Abstract Syntax Tree Apr 15
11 Traverse Tree for Code Generation Apr 22
12 Test Hello World Apr 26
13 Complete Code Generation for added features Apr 30
14 Debugging and Regression Tests May 3
15 Complete all modules May 8
16 Documentation May 10
Final Report Complete Today
Tab+Q Quick Add Tab+BKSP Delete Task Ctrl+↓ Move Down Ctrl+↵ Mark Complete more...
Share Asana

Language Evolution

Tanay Jaipuria

Based on the feedback we received on the Language Proposal and the Language Reference Manual, we knew that it would be nearly impossible to implement all the features we had envisioned MineTime to have.

Therefore we first ranked all the features in three buckets—those that were essential, those that we really should try to include and those that were not really needed.

We then got rid of the features we agreed to be unnecessary. These included a few built-in functions such as tower and transpose and complex data-types such as struct.

The features we had listed as essential were: easy creation of the map type, if/else statements, while loops, function declarations and function calls.

The features we had listed as useful but low-priority: for loops (since they are equivalent in power to while loops), lists, and classes.

As Melanie suggested, we decided to employ a breadth-first approach. Our first goal was to get hello world working completely. Then we updated the grammar and the traversal to add selection statements. Afterwards, we implemented for loops, function calls, and so on. That way, we would always have a working compiler.

As we continued to build our language, we eliminated classes.

Compiler Tools

Given the programming backgrounds of our team members, we settled on Python as the target language for our compiler.

After conducting extensive research, we decided to use PLY[1] (Python Lex-Yacc) for our Lexer and parser. It was simple to use and had good documentation and numerous examples that helped us get up and running.

We wrote the tree traversal module in-house.

Libraries

We used one external library in our compiler: pymclevel[2]. It is a library written in Python that makes it easier to build Minecraft levels. Our compiler translates the input program written in

[1] "PLY (Python Lex-Yacc)." 2005. 11 May. 2013 <http://www.dabeaz.com/ply/>
[2] "mcedit/pymclevel · GitHub." 2012. 11 May. 2013 <https://github.com/mcedit/pymclevel>

MineTime to an output Python program. It imports and uses the pymclevel library to create maps and add blocks to the map. In addition, pymclevel uses the libraries Numpy and PyYaml.

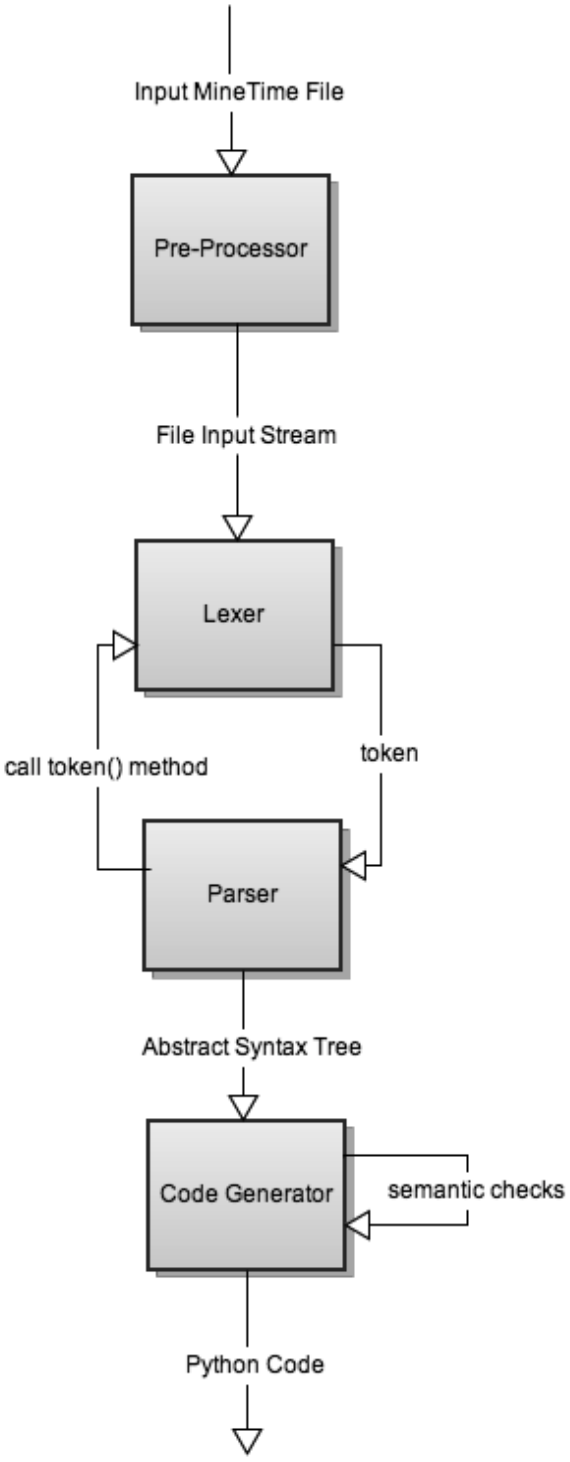
Consistency

After receiving feedback on the LRM and prior to working on our compiler, we updated our LRM to match the new specifications of our language. For example, we removed the built-in functions tower and transpose. We then used the LRM to list out all the features our language had and ranked them after placing them into buckets, implementing them one by one. This way, we made sure that we knew about everything in the LRM and that the feature we implement already existed in the LRM.

Sometimes, we had to deviate from the LRM. For example, we found that we had to modify the grammar to fix conflicts we encountered. We also made additions to Lex and Yacc in the form of new reserved keywords or modified syntactic constructs. In both cases, we made sure to update the LRM so consistency was always preserved.

Translator Architecture

Don Yu



Pre-Processor

At the first step, we take the entire MineTime input file and preprocess it for imports. The input here will be the actual MineTime file the user wants to compile and the output of the Pre-Processor will be a File Input Stream that results. The Pre-Processor was written by Don.

Lexer

After preprocessing, the string contents of the input file are passed to the Lexer, which continuously matches the string contents with Token specifications. The initial Lexer was written by Tanay, but all members of the team contributed to the final version of the Lexer.

Parser

The Parser contains our grammar and communicates with the Lexer to continuously get tokens as needed. Here a token is a class LexToken that contains a type (the token class) and a value (the string that resulted in this). As the Parser evaluates the tokens, it is also creating an AST. This AST comes in the form of Nodes that contain children, a type (i.e. the non-terminal or terminal name), and a leaf value. The Parser will output the root Node of the AST. Armaan wrote the class definition for an AST Node and the entire team contributed to creating the parser, with Stephen responsible for ensuring that the parser grammar matched our LRM.

Code Generator

The AST from the Parser is then passed on to the last step, the Code Generator, which we also combined with the Semantic Analyzer. Here we traverse the tree, run semantic type-checking and scope-checking, and then create the Python code. The Code Generator process was initially written by Tanay, our Language Guru, and the semantic analysis parts were written by Don. Other team members contributed to the final version of the Code Generator (i.e. fixing things or adding a code creation mechanism for new code features). The resultant Python code is captured as a string and written to a file that can then be executed by the user.

All of these modules are brought together and the whole process is run through our wrapper module minetime.py, written by our System Integrator Patrice.

Development and Runtime Environment

Patrice Liang

The compiler was developed under a Unix-based environment, specifically Ubuntu and Mac OS. All members used vim and the Terminal to edit and test code. Git and Github were used for version control and source-code sharing.

As we used Python 2.7.2 to create and test our compiler, a generic Makefile was not necessary. Instead, our file minetime.py takes in as argument the user-defined .mt file containing MineTime code and outputs the corresponding .py file that can be executed to create a map.

Minetime.py:

```
import yaccing as yacc
import sys
from lexing import Mtlex
from traverse import *
from preprocess import *

def main(argv):
    inputfile = argv[1]
    filename = inputfile.split(".")[0]
    source = open(inputfile).read()
# generate the parser
    parser = yacc.getyacc()
# generate the Lexer to be used with parser
    m = Mtlex()
    m.build()
# preprocessing step
    preprocessor = Processor()
    source = preprocessor.preprocess(source)
    tree = parser.parse(source, lexer=m.lexer)

    firstline = '''import logging
import os
import sys
from pymclevel import mclevel
from pymclevel.box import BoundingBox
'''
    lastline = '''
if __name__ == '__main__':
     main()
'''
```

```
    result = Traverse(tree).getpython()
    code = firstline + "\n" + result + "\n" + lastline + "\n"
    outputfile = filename + ".py"
    output = open(outputfile, 'w')
    output.write(code)

if __name__ == '__main__':
    main(sys.argv)
```

The compiler is run in a virtual environment to avoid conflicts with existing Python files. Command-line instructions to set up the virtual environment and install all requirements are as follows:

```
pip install virtualenv
virtualenv env
source env/bin/activate
pip install pyyaml
pip install ply
pip install numpy
```

Test plan

Stephen Zhou

Tools

Testing of the compiler relies on Python's unittest module in the standard library. Each part of the compiler, lex, yac, and traverse, has its own set of unittests. Moreover, there is a unittest for the external pymclevel library as a way for a group member to quickly check if he or she has correctly installed pymclevel.

All testing harnesses are found within the tests/ directory of the project's root. Each test suite is listed and described as follows.

Relevant Files and Descriptions

`tests/templevel.py`

templevel.TempLevel takes in a filename and creates an empty Minecraft map. It uses the atextit and shutil modules to delete itself upon finishing the test.

`tests/mt_test_cases.py`

mt program test snippets that are used for testing the yacc and traverse are found within tests/mt_test_cases.MTTests.

`tests/level_test.py`

level_test.TestLevelCreation uses templevel.TempLevel to create an empty Minecraft map, fill it with blocks, and checks whether or not the blocks exist in the game with assertions.

`tests/lexing_test.py`

lexing_test.TestLexing uses several equality assertions to check the correctness of lexing.py.

Test cases were created to test general code and potential edge cases.

`tests/yaccing_test.py`

yaccing_test.TestYaccing does not do any direct testing of the parse tree generated by yaccing.py. Instead, each test case prints out the tree generated by the cases found in mt_test_cases.MTTests and is manually inspected by a group member.

Equality assertion tests were not made because of the complexity of checking for equality of large parse trees. However, there tests/make_trees.py automatically generates and writes the yaccing output of each test case into their individual files in tests/testfiles/.

`tests/traverse_test.py`

traverse_test.TestTraverse prints out the translated Python code of mt_test_cases.MTTests cases.

Like yaccing_test, traverse_test has few assertions. Code is inspected and run manually.

Select Test Cases

Selected test cases are as follows. A complete collection of test cases are found in tests/mt_test_cases.py

```
helloworld = """
def main() {
    x = new flatmap("testfilesgitestmap",   500,500,500);
    b = new Point(10,20,30);
    x.add(block(STONE), b);
    x.close();
}"""

if_elseif_else = """
def main() {
    i = 2;
    if (i == 1)
       i = 2;
    else if (i==2) {
       i = 3;
    } else
       i = 4;
}"""

empty_function = """
def main(1,2) {
}"""

relations_arithmetic = """
def main() {
    a && b;
    a || b;
    a == b;
    a != b;
    a > b;
    a < b;
    a >= b;
    a <= b;
```

```
    a + b;
    2 - 3;
    3 * 3;
}"""

make_blocks = '''
def makeblocks(start, end, x) {
    while (start < end) {
        c = new Point(0,0,start);
        x.add(block(COBBLESTONE), c);
        start = start + 1;
    }
    for (;start<end;start=start+1)
    {
        c = new Point(0,0,start);
        x.add(block(COBBLESTONE), c);
    }
}

def main() {
x = new Flatmap("testfiles/testmap",500,500,500);
makeblocks(0,10, x);
x.close();
}
'''

melanie = '''
def even(x) {
    answer = false;
    div2 = x/2;
    times2 = div2*2;
    if (times2 == x) {
        answer = true;
    }
    else {
        answer = false;
    }
    return answer;
}

def main() {
    map = new Flatmap("xmap", 300, 300, 100);

    for (x=0; x < 300; x = x+1) {
        for (y=0; y < 300; y = y+1) {
```

```
            if ( even(x) ) {
                p = new Point(x,y,0);
                map.add(block(COBBLESTONE), p);
            }
            else {
                p = new Point(x,y,0);
                map.add(block(BRICK), p);
            }
        }
    }

    for (x=0; x < 300; x = x+1) {
        for (y=300; y > 0; y = y-1) {
            if ( even(x) ) {
                p = new Point(x,y,0);
                map.add(block(COBBLESTONE), p);
            }
            else {
                p = new Point(x,y,0);
                map.add(block(BRICK), p);
            }
        }
    }

    map.close();
}
'''
```

Notable Bugs Encountered

1. Empty functions, statements, etc. translated to Python initially had a hanging indent (e.g. if i == 1:), which does not run. Must put a pass in the Python code upon translation.
2. Checking for existence of tree leaves using if self.leaf evaluates all strings and ints to True except for 0, which evaluates to false which messed up type-checking for very select cases.
3. Main method in MineTime not translating to Python, so no code runs.

Conclusions

Lessons learned as a team

1. A great language idea is important, but at one point it becomes more important to just settle on one idea, get started, and hone out the details later. This is particularly important for a semester-long project like such.
2. The duration of meetings is nowhere as important as their productivity. Even if we could only meet for an hour or two on some occasions, the momentum from each of these focused and productive meetings carried through to personal work time and the next meeting.
3. It's fine to cut down. Though it may be hard to see a feature envisioned in the beginning go, ultimately it's more important to prioritize and first and foremost focus on building a working compiler.

Lessons learned Individually

Mirza Armaan

Building a project from scratch, isn't as easy as it seems. From the initial design phase till the final completion stage we learnt to go back to the drawing board, look at what we had and come up with an innovative way to move forward. Working in a team and collaborating effectively were one of the key lessons learnt during this course.

1. Be organized, and keep everyone up to date with weekly tasks.
2. Have a good design and understanding of your project before you start working. In all likelihood there will be modifications to your design along the way and making changes will be easy if you have a clear view of your final goal.
3. Teamwork always boosts productivity. Working on the project together always allows for creative exchange of ideas and this helps push the team forward.
4. A brief yet descriptive explanation of code check-ins helps everyone be on the same page. Keep your GitHub commits short and sweet.
5. An interesting quote which summarizes the implementation of our project:
   " Without requirements or design, programming is the art of adding bugs to an empty text file. "
   -Louis Srygley

Tanay Jaipuria

MineTime was my first large-scale group programming project at Columbia, and I greatly enjoyed it. I learned a lot about how compilers worked and about working in teams. Below are some my key takeaways from the project:

1. A good group makes things a lot simpler and a lot more fun.

2. No matter how early you start, you will be working on your project right up to the deadline.
3. Version control is a lifesaver - it makes programming in groups much easier
4. There are certain synergies that take place when everyone in the group is sitting and working together. If I were to estimate, I would say that we got at least twice the amount of work done when working together in a group for two hours than we did when we each worked two hours separately. In other words, the team as a whole is more efficient than the sum of its parts.
5. As Dwight Eisenhower once said, “Plans are worthless, but planning is everything.” We tried to stay very organised and planned a lot, using asana to assign tasks and deadlines and to divide up the work. Although deadlines were not always met and tasks were not always done in the order we planned to do them in - the very act of planning helped us chart out how much we had done and how much we had left and to ensure that we had a working compiler by the deadline.

Don Yu

From the first day I picked up programming, there had always been nuances of programming languages that I just didn’t understand. All those past grievances came to a culmination this semester as I started to understand that every nuance of a language was meant to serve a purpose, whether it be to help with type-checking, scope-checking, or make the programmers of the language compiler-writer happy. For example, while implementing type checking for functions in MineTime, I realized that it was impossible to type-check a function before it was even defined by the user. So that was why C forces you to declare or define functions before you use them. Even more important were all of the programming tips I learned.

1. Don’t take a shortcut or shrug off a poorly-written block of code in an attempt at speed. They will come back to haunt you.
2. Python has a lot of features and if you think you’re writing too much code to get something done, then you probably are
3. Don’t underestimate the time it takes to get a working development environment up and running as some python libraries don’t always work the same on different environments
4. It’s okay to start with just one big file, but you have to know when it’s time to start breaking it up into modules that individual members can work on.
5. Always have a designated tester for any programming team, even if every programmer on the team plans to write test-cases and run tests himself or herself. The tester will still find bugs for use-cases and scenarios that the code creator didn’t foresee.

Patrice Liang

Applying the topics we learned about in class to our project was extremely rewarding, and it gave me a completely different understanding of the material. To me, though, an even more valuable experience was working in a group and having everyone invested in the same goal/product.

1. Roles are good. Although each team member was not limited to tasks associated with his/her role, I still found it important and useful to have official roles assigned. It was very productive to have a go-to person for different concerns, and each member felt a greater sense of responsibility for his/her scope. Furthermore, since our roles were actually randomly selected at the outset, I was able to learn more about and work in an area I would not have otherwise been exposed to.
2. Packaging and delivery are just as important as the product. In looking at past Language White Papers, Language Reference Manuals, and Language Tutorials as well as writing our own, I realized the importance of being able to effectively present a product and communicate its specifications. Without clear communication of functionalities and appeal, a language will be of little use and interest to the third party, however intricate.
3. Face-to-face meetings should not be overlooked. Compared to certain other tasks, programming is generally considered to be more solitary, as members of a team can essentially work on a project anywhere given a laptop and an Internet connection. However, through this project I have learned further that in-person meetings are absolutely essential, as it not only improves communication and efficiency, but also boosts camaraderie and productivity.

Stephen Zhou

I learned a lot from my first significant group programming project at Columbia.

1. Tests are surprisingly hard to design. In the end, I ended up just hooking most of the unittests up to print statements rather than assertions because the time it takes to design and create good, persistent test cases is overkill for the time scope of this project.
2. One small, seemingly innocent change can break everything. Adding or moving around productions in the grammar can completely mess up tree traversals.
3. GitHub is bigger and better than life. Version control is a life-saver.
4. Translating to Python without static typing is very difficult.
5. Regex search and replace is awesome. It makes refactoring and working with large blocks of text a breeze.
6. No matter how much I think I know about Python, there's always more to learn.
7. Comments are important, as many times I had a hard time understanding what other people were doing.

Advice for future teams

1. Start early.
2. Use version control - the hour or so it takes to setup Github is most definitely worth it.
3. Don't dilly-dally with language design decisions. Make a decision with the group quickly and stick with it.
4. Do the majority of your planning before starting on the code.

5. Work on your code in a breadth first manner. We adopted this approach and this ensured that we always had some sort of working compiler to fall back on in case calamity struck.
6. Meet the TA's and Professor often to get feedback. Take that feedback into account and work on your language further. Repeat.

Suggestions

1. It would be great to have milestones every few weeks in terms of where your group should be at any point in time throughout the course. For our group, it was very hard to determine how long a task would take and/or how far ahead we were or weren't. Milestones (not to be turned in but just as a guideline) would help all groups measure their progress and keep on track.
2. The homework assignments helped us a great deal, especially in regards to working with Lex and Yacc. However, we struggled more with figuring out how to implement type checking and scoping. Particularly, this was a hard task to do when converting to an interpreted language like Python. It would be very helpful to perhaps include more coverage of this in the syllabus and also some implementation in homework assignments.

Appendix

Git Log

a3705c2	Stephen Zhou	2 hours ago	organization stuff and forgot to add traverse_test
b3c88b2	Stephen Zhou	2 hours ago	Merge branch 'master' of github.com:donyu/minetime
630ff5d	Stephen Zhou	2 hours ago	yaccing test revamp
432324f	Tanay Jaipuria	2 hours ago	Merge branch 'master' of https://github.com/donyu/minetime
f4b426c	Tanay Jaipuria	2 hours ago	Added pass for if/else/loops/functions
c5dca0f	Don Yu	2 hours ago	added in checking to see if function or initializer was defined before
56723fe	Don Yu	13 hours ago	Merge branch 'master' of github.com:donyu/minetime
e6e6fe2	Don Yu	13 hours ago	type checking for user-defined functions complete
7e42376	Stephen Zhou	13 hours ago	merged
925d9b0	Stephen Zhou	13 hours ago	commenting and delete print
0785c23	Don Yu	13 hours ago	Merge branch 'master' of github.com:donyu/minetime
34a813c	Don Yu	13 hours ago	Commenting out points in Lex because we use initializer now
02ce76a	Patrice Liang	14 hours ago	minetime.py outputs executable .py file
07672d1	Patrice Liang	14 hours ago	Merge branch 'master' of github.com:donyu/minetime
f155e24	Patrice Liang	15 hours ago	Takes in .mt file and outputs .py file
efbca6c	Don Yu	15 hours ago	making a point with the new data code example in yaccing
84c6de6	Don Yu	15 hours ago	fixed block exception so that correct block type exception is thrown
3cb16ea	Don Yu	16 hours ago	changing fill method so that we indent if within curly braces
d06330f	Don Yu	16 hours ago	now traverse throws an error when using variable outside of something's scope
37f107d	Don Yu	17 hours ago	added in scope checking by curly braces
9085751	Don Yu	20 hours ago	fixed traverse to not remove semicolons
ca825de	Stephen Zhou	20 hours ago	Merge branch 'master' of github.com:donyu/minetime
16fefce	Stephen Zhou	20 hours ago	added all block types
00f465b	Don Yu	20 hours ago	need to throw a compiler error if importing non-existant file
c8444c2	Don Yu	20 hours ago	merging with last push
b4b5051	Don Yu	20 hours ago	added in preprocessing for imports
c957ed0	Tanay Jaipuria	20 hours ago	Added support for true and false in lex
03874dc	Armaan	21 hours ago	Added traverse function for FOR LOOP
cf6c560	Stephen Zhou	23 hours ago	update grammar to match lrm
b580add	Stephen Zhou	3 days ago	remove grammar redundancy and added comments

7a0d223	Stephen Zhou	3 days ago	better yacc error messages
6274f6f	Don Yu	3 days ago	more extensible way to check initializer arguments (types and number of)
dd66584	Tanay Jaipuria	4 days ago	Updated readme with tasks - someone has to do type checking and scoping and then we're in a good place
135d415	Tanay Jaipuria	4 days ago	Works for multiple functions
7554891	Armaan	4 days ago	Merge branch 'master' of https://github.com/donyu/minetime
78c3ca2	Armaan	4 days ago	Updated traverse to handle new grammar. Multiple Functions NOT implemented yet
2062268	Patrice Liang	4 days ago	Removed redundant reserved keyword in dict
c421888	Armaan	4 days ago	Added Stephens update to grammar.txt
67cd737	Stephen Zhou	5 days ago	did grammar changes in readme. NOTE: added new keyword for objects to eliminate reduce/reduce conflict
46d14ca	Armaan	5 days ago	Updated Readme with tasks
41beee4	Tanay Jaipuria	5 days ago	Updated Readme
5fc357c	Tanay Jaipuria	5 days ago	Updated Grammar to allow for map.add etc to be called within it and also within for loops. Also updated readme.
4b6388d	Tanay Jaipuria	5 days ago	Merge branch 'master' of https://github.com/donyu/minetime
3c286dc	Tanay Jaipuria	5 days ago	Handled a bit of stuff for flatmap and allowed for params for functions
41672c1	Armaan	5 days ago	Added while implementation
1.64E+12	Armaan	6 days ago	Merge branch 'master' of https://github.com/donyu/minetime
347bdc7	Armaan	6 days ago	Handling if control flow
8ac3149	Tanay Jaipuria	6 days ago	Statements within if and else kinda work now...
4f7ba41	Armaan	6 days ago	Merged with Tanay
577244	Armaan	6 days ago	Added operations for if control flow
084bcab	Tanay Jaipuria	6 days ago	Taking care of points function so that variables can be used within a point
099ad49	Tanay Jaipuria	6 days ago	works for Point(a
02de1ae	Tanay Jaipuria	6 days ago	Merge branch 'master' of https://github.com/donyu/minetime
12d5cba	Tanay Jaipuria	6 days ago	Hello world works again yayyy
9eb5a50	Stephen Zhou	6 days ago	small correction
063e6d7	Stephen Zhou	6 days ago	formatted grammar
54644c0	Stephen Zhou	6 days ago	oops forgot I changed lexing too
f2940bf	Stephen Zhou	6 days ago	almost complete grammar
37c170d	Stephen Zhou	6 days ago	added function definitions and changed test cases to reflect new grammar
1.12E+10	Don Yu	6 days ago	using traverse.py now
b06da5b	Don Yu	6 days ago	renaming because we don't need traverse and NewTraverse
7848fe2	Stephen Zhou	11 days ago	added grammar and tests for for loop and if elif else

c6100ef	Stephen Zhou	11 days ago	Merge branch 'master' of github.com:donyu/minetime
f82066c	Stephen Zhou	11 days ago	added for
f274f83	Tanay Jaipuria	11 days ago	Added > and < to LEX
e575eae	Tanay Jaipuria	13 days ago	Changed grammar slightly and added functionality to initialize points and use those in add function
cac670f	Stephen Zhou	13 days ago	more testing instructions. please read
cb39bbe	Stephen Zhou	13 days ago	removed lex_yacc folder (unnecessary)
5fca678	Stephen Zhou	13 days ago	readme updated to reflect proper
99bd308	Stephen Zhou	13 days ago	yaccing tests commented out -- run yaccing_test.py in tests/ for test modular test suite
a50ac4b	Stephen Zhou	13 days ago	yaccing-extended now yaccing. old yaccing versions 1
e1dc7c9	Stephen Zhou	13 days ago	redid some tests
cb347f8	Tanay Jaipuria	13 days ago	Sorry Stephen
d39091f	Tanay Jaipuria	13 days ago	Got code working when given as one string
739d5e6	Stephen Zhou	13 days ago	renaming and moving files around
eaf3e0f	Tanay Jaipuria	2 weeks ago	Added new traverse to work for hello world with yaccing extended
98f44e4	Stephen Zhou	2 weeks ago	Merge branch 'master' of https://github.com/donyu/minetime
f8d91c7	Stephen Zhou	2 weeks ago	and traverse...
64b28e0	Stephen Zhou	2 weeks ago	big grammar revamp
64b838b	Tanay Jaipuria	2 weeks ago	Changed grammar and moved point to primary expression and got traverse to work with new grammar
292754a	Tanay Jaipuria	2 weeks ago	Added error checking for blocks
57a0f13	Stephen Zhou	2 weeks ago	some minor changes + blockid textfile
f293476	Tanay Jaipuria	2 weeks ago	Added error checking for add and block
e0b90a3	Tanay Jaipuria	2 weeks ago	Added error handling for blocks
55aaf71	Tanay Jaipuria	2 weeks ago	Added error handling for blocks
2962d4d	Tanay Jaipuria	2 weeks ago	Simplified code for hello world
5c641dc	Tanay Jaipuria	2 weeks ago	Creates a python file
c0f439e	Stephen Zhou	2 weeks ago	merge style fixes
db16874	Stephen Zhou	2 weeks ago	small style changes
d43be19	Tanay Jaipuria	2 weeks ago	Merged traverse
c784afe	Tanay Jaipuria	2 weeks ago	Hello world kinda working
81f355e	Stephen Zhou	2 weeks ago	helloworld2
d8e9069	Armaan	2 weeks ago	Added translation for add function
3a49e5a	Tanay Jaipuria	2 weeks ago	Added translation for blocks to traverse

0f5edea	Tanay Jaipuria	2 weeks ago	Added stuff
b646c59	Tanay Jaipuria	2 weeks ago	Merge branch 'master' of https://github.com/donyu/minetime
efcd742	Tanay Jaipuria	2 weeks ago	Changed traverse
15b3927	Stephen Zhou	2 weeks ago	rename vars
049172a	Stephen Zhou	2 weeks ago	Merge branch 'master' of github.com:donyu/minetime
f5c5e8e	Stephen Zhou	2 weeks ago	merge fix
2e6ed90	Stephen Zhou	2 weeks ago	tostr
3122fdc	Tanay Jaipuria	2 weeks ago	Added functions to traverse
64caf12	Tanay Jaipuria	2 weeks ago	Merge branch 'master' of https://github.com/donyu/minetime
16703a8	Tanay Jaipuria	2 weeks ago	Added traverse link
b1154bb	Stephen Zhou	2 weeks ago	some more testing hookups for lex
03f3544	Tanay Jaipuria	2 weeks ago	Updated ast again
1cf777f	Tanay Jaipuria	2 weeks ago	Updated ast
5311c57	Tanay Jaipuria	2 weeks ago	Added code for AST
4535ae7	Don Yu	2 weeks ago	indentation was weird cause I wasn't using spaces
4122063	Don Yu	2 weeks ago	Merge branch 'master' of github.com:donyu/minetime
22da353	Don Yu	2 weeks ago	grammar for hello world program complete
b920634	Stephen Zhou	2 weeks ago	lexing point regex and test
edcae84	Stephen Zhou	2 weeks ago	lexing test
96d6ff4	Don Yu	2 weeks ago	grammar for line #1 complete
399823	Don Yu	2 weeks ago	adding grammar rules for 1st line
30c135c	Don Yu	2 weeks ago	Merge branch 'master' of github.com:donyu/minetime
9ca56d2	Stephen Zhou	2 weeks ago	tests
73cda71	Don Yu	2 weeks ago	merge with tanay
f8ec3d2	Don Yu	2 weeks ago	need semicolons
3195ad0	Tanay Jaipuria	2 weeks ago	Updated lex to handle functions
503fb2f	Don Yu	2 weeks ago	making lexing easier to use at least for me
07056f7	Stephen Zhou	3 weeks ago	clean up dir
84ad120	Stephen Zhou	3 weeks ago	readme formatting
7b7870a	Stephen Zhou	3 weeks ago	adding testing portion to readme and commenting in tests
5e7e597	Stephen Zhou	3 weeks ago	JESUS CHRIST
44e9509	Stephen Zhou	3 weeks ago	Added __init__.pys for modularity
c1db706	Stephen Zhou	3 weeks ago	seperating code sandbox and actual unittests
53f5725	Tanay Jaipuria	3 weeks ago	Added functionality to lex

a8c9d3e	Stephen Zhou	4 weeks ago	unnitest
67cb91f	Stephen Zhou	4 weeks ago	added setup.py link to readme
fc406c9	Don Yu	4 weeks ago	tutorial for lex_yacc
4e432eb	Don Yu	4 weeks ago	added back in lex_yacc folder
06c7fcd	Don Yu	4 weeks ago	adding dot operator to lexing
0c29e05	Don Yu	4 weeks ago	merging old repo with new
25bb378	Stephen Zhou	4 weeks ago	Merge branch 'master' of github.com:donyu/minetime
655e8dd	Stephen Zhou	4 weeks ago	changelog added
ee7c417	stepzhou	4 weeks ago	Readme formatting
ddb167c	Tanay Jaipuria	4 weeks ago	Adding lexing file
28b6154	Tanay Jaipuria	4 weeks ago	Merge branch 'master' of https://github.com/donyu/minetime
fd5c857	Stephen Zhou	4 weeks ago	basic hello world
e92b648	Stephen Zhou	4 weeks ago	new site-packages
835e2e6	Stephen Zhou	4 weeks ago	rm site-packages
6c56987	Stephen Zhou	4 weeks ago	mcedit site-packages
e842220	Stephen Zhou	4 weeks ago	remove gitignores
ac713c6	Don Yu	4 weeks ago	pip install rather than easy_install numpy
9ddf8a8	Don Yu	4 weeks ago	git clone pymclevel
bbc12ba	Don Yu	4 weeks ago	update pip install virtualenv
f076f5d	Don Yu	4 weeks ago	Merge branch 'master' of github.com:donyu/minetime
727ba3b	Don Yu	4 weeks ago	dont want these files anymore cause ply can be pip installed
d63e033	Don Yu	4 weeks ago	dont want these files anymore cause ply can be pip installed
428a782	Don Yu	4 weeks ago	update README
7b441e4	Don Yu	4 weeks ago	trying virtual environment again
972659c	Don Yu	4 weeks ago	delete this site-packages from repo
cf8c0c3	Don Yu	4 weeks ago	merging
9206ca5	Don Yu	4 weeks ago	want to override this
d9c0201	Don Yu	4 weeks ago	Initial commit
285727c	Stephen Zhou	4 weeks ago	site-packages
4df4e99	Stephen Zhou	4 weeks ago	only track site-packages
32612ae	Stephen Zhou	4 weeks ago	pymclevel no longer submodule
ca8cb9d	Stephen Zhou	4 weeks ago	env
b92c208	Stephen Zhou	4 weeks ago	tests
d5fbb06	Don Yu	4 weeks ago	Merge branch 'master' of github.com:donyu/minetime

02059c0	Don Yu	4 weeks ago	using virtual env so dont need this
f64cb2a	armaan110	4 weeks ago	Update README.md
8193471	armaan110	4 weeks ago	Create README.md
a7f387a	Don Yu	4 weeks ago	lex yacc python
254da60	Don Yu	4 weeks ago	added in lexer and yacc python modules
bdf9a73	Don Yu	4 weeks ago	do we want pymclevel included always

Source Code

Tests

level_test.py

```
import sys
import unittest
import filecmp
from filecmp import dircmp

sys.path.append('..')
from pymclevel.box import BoundingBox
import templevel

STANDARD = 'testfiles/Standard'

class TestLevelCreation(unittest.TestCase):

    def setUp(self):
        self.testlevel = templevel.TempLevel("Temp")

    def test_chunk_creation(self):
        level = self.testlevel.level

        level.createChunk(0, 0)
        level.saveInPlace()
        self.assertTrue(level.containsChunk(0, 0))

    def test_fill(self):
        level = self.testlevel.level

        level.createChunk(0, 0)
        cx, cz = level.allChunks.next()
        box = BoundingBox((cx * 16, 0, cz * 16), (32, level.Height,
32))
        level.fillBlocks(box, level.materials.WoodPlanks)
        level.fillBlocks(box, level.materials.WoodPlanks,
[level.materials.Stone])
        level.saveInPlace()
        c = level.getChunk(cx, cz)

        assert (c.Blocks == 5).all()

if __name__ == "__main__":
    unittest.main(verbosity=2)
```

lexing_test.py

```
import unittest
import sys
```

```
from cStringIO import StringIO
from itertools import izip

sys.path.append('..')
import lexing

class TestLexingSyntax(unittest.TestCase):

    def setUp(self):
        self.lex = lexing.Mtlex()
        self.lex.build()
        self.lexer = self.lex.lexer

    def test_tokens(self):
        """
        Checks the correctness of single tokens
        """
        cases = {1 : 'NUMBER',
                 12345 : 'NUMBER',
                 '+' : 'PLUS',
                 '-' : 'MINUS',
                 '/' : 'DIVIDE',
                 '(' : 'LPAREN',
                 ')' : 'RPAREN',
                 'but34' : 'ID',
                 '=' : 'ASSIGN',
                 '{' : 'LCURL',
                 '}' : 'RCURL',
                 '"H3#;.LLO WORLD"' :'STRING',
                 '# #h32.ello' : 'COMMENT',
                 ',' : 'COMMA',
                 '(  1, 2,  3)' : 'POINT',
                 ';' : 'SEMICOLON',
                 ':' : 'COLON'}
        self.assert_tokens_eq(cases)

    def assert_tokens_eq(self, cases):
        """
        Takes in dictionary of value inputs and checks for type and
value
        correctness of the LexToken output
        """
        for key, val in cases.iteritems():
            self.lexer.input(str(key))
            token  = self.lexer.token()
            self.assertEqual(key, token.value)
            self.assertEqual(val, token.type)

    def test_helloworld(self):
        """
        Checks the correctness of helloworld.mt
        """
        progfile = open('testfiles/helloworld.mt', 'r')
```

```
        expfile = open('testfiles/helloworld.out', 'r')

        self.assert_prog(progfile, expfile)

    def test_sandbox(self):
        """
        Sandbox for mt program lex output
        """
        progfile = open('testfiles/sandbox.mt', 'r')
        print self.lex.tok_str(progfile.read())

    def assert_prog(self, progfile, expfile):
        """
        Takes in a mt programming file and expected output file and
checks for
        correctness
        """
        tokens = StringIO(self.lex.tok_str(progfile.read()))

        for t, o in izip(tokens, expfile):
            self.assertEqual(t, o)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

make_trees.py

```
import re
import sys
from mt_test_cases import MTTests

sys.path.append('..')
import yaccing

parser = yaccing.parser

for attr in MTTests.__dict__.keys():
    if not re.match(r'^__.*__$', attr):
        result = parser.parse(getattr(MTTests, attr))
        f = open('testfiles/' + attr, 'w')
        f.write(result.__str__())
```

mt_test_cases.py

```
class MTTests(object):
    """
    Wrapper class for yaccing and traverse test cases. These are mt
code
    snippets
    """
```

```
    helloworld = """
def main() {
    x = new flatmap("testfilesgitestmap",   500,500,500);
    b = new Point(10,20,30);
    x.add(block(STONE), b);
    x.close();
}"""

    compound = """
def main() {
    { i=0;i=1;i=2; }
}"""

    while_loop = """
def main() {
    while (i=1) {i=1;i=1;i=1;}
    i = 0;
}"""
    for_loop = """
def main() {
    for (i = 1; i = 1; i = 1) {
        i = 1;
    }
}"""
    if_stmt = """
def main() {
    if (i=222220) {}
}"""

    if_else = """
def main() {
    i = 2;
    if (i = 1)
       i = 2;
    else {
       i = 3;
    }
}"""
    if_elseif_else = """
def main() {
    i = 2;
    if (i == 1)
       i = 2;
    else if (i==2) {
       i = 3;
    } else
       i = 4;
}"""

    zero_bug = """
def main() {
    i = 0;
```

```
    i = 1;
}"""
    relations_arithmetic = """
def main() {
    a && b;
    a || b;
    a == b;
    a != b;
    a > b;
    a < b;
    a >= b;
    a <= b;
    a + b;
    2 - 3;
    3 * 3;
}"""
    empty_function = """
def main(1,2) {
}"""
    complicated = """
def main() {
    a * (b - 3) + 3 || 5;
}"""
    external = """
i = 1;

def f(1, 2) {
}

def main() {
    if (a > 3) {
        i;
    }
}"""

    return_stmt = """
def main() {
    i = 1;
    return 1;
    return;
}"""

    assignment = """
def main() {
    i = a + 2;
    j = 3 < 2;
}"""

    make_blocks = '''
def makeblocks(start, end, x) {
    while (start < end) {
        c = new Point(0,0,start);
        x.add(block(COBBLESTONE), c);
```

```
        start = start + 1;
    }
    for (;start<end;start=start+1)
    {
        c = new Point(0,0,start);
        x.add(block(COBBLESTONE), c);
    }
}

def main() {
x = new Flatmap("testfiles/testmap",500,500,500);
makeblocks(0,"hi", x);
x.close();
}
'''

    add_block = '''
def main() {
    a = 2;
    if (a > 1) {
        b = 2;
    }
    x = new Flatmap("testfiles/testmap", 500, 500, 500);
    c = new Point(0, 0, 0);
    x.add(block(STONE), c);
    x.close();
}
'''

    def __init__(self):
        pass
```

runtests.py

```
import unittest
import sys
import os

if __name__ == "__main__":
    all_tests = unittest.TestLoader().discover('.',
pattern="*_test.py")
    unittest.TextTestRunner(verbosity=2).run(all_tests)
```

templevel.py

```
import atexit
import os
from os.path import join
import shutil
from pymclevel import mclevel

TEST_DIR = 'testfiles/'
```

```
class TempLevel(object):

    def __init__(self, filename, delete=True):
        self.filename = filename
        self.testpath = join(TEST_DIR, filename)

        if os.path.exists(self.testpath) and delete:
            shutil.rmtree(self.testpath)

        self.level = mclevel.MCInfdevOldLevel(self.testpath,
create=True)
        atexit.register(self.removeTemp)

    def removeTemp(self):
        if os.path.isdir(self.testpath):
            shutil.rmtree(self.testpath)
```

traverse_test.py

```
import unittest
import sys
import ply.yacc as yacc
from mt_test_cases import MTTests

sys.path.append('..')
import yaccing
from traverse import Traverse

class TestTraverse(unittest.TestCase):
    """
    Tests the traverse for correctness
    """

    firstline = '''
import logging
import os
import sys
from pymclevel import mclevel
from pymclevel.box import BoundingBox'''

    def setUp(self):
        self.parser = yaccing.parser

    def test_helloworld(self):
        self.traverse(MTTests.helloworld)

    def test_compound(self):
        self.traverse(MTTests.compound)

    def test_while_loop(self):
        self.traverse(MTTests.while_loop)
```

```
    def test_if(self):
        self.traverse(MTTests.if_stmt)

    def test_if_elseif_else(self):
        self.traverse(MTTests.if_elseif_else)

    def test_make_blocks(self):
        """
        Check for type mismatch
        """
        with self.assertRaises(Exception):
            self.traverse(MTTests.make_blocks)

    def test_add_block(self):
        self.traverse(MTTests.add_block)

    def test_empty_function(self):
        self.traverse(MTTests.empty_function)

    def traverse(self, prog):
        result = self.parser.parse(prog)
        translated = Traverse(result).getpython()
        code = "\n{0}\n{1}\n\n".format(self.firstline, translated)
        print code


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

yaccing_test.py

```
import unittest
import sys
import ply.yacc as yacc
from mt_test_cases import MTTests
from os.path import join

sys.path.append('..')
import yaccing
from textwrap import dedent
from lexing import Mtlex

test_dir = 'testfiles'

class TestYaccing(unittest.TestCase):

    def setUp(self):
        self.parser = yaccing.parser

    def test_helloworld(self):
```

```
        self.print_result(MTTests.helloworld)

    def test_compound(self):
        self.print_result(MTTests.compound)

    def test_while(self):
        self.print_result(MTTests.while_loop)

    def test_for(self):
        self.print_result(MTTests.for_loop)

    def test_if(self):
        self.print_result(MTTests.if_stmt)

    def test_if_else(self):
        self.print_result(MTTests.if_else)

    def test_if_elseif_else(self):
        self.print_result(MTTests.if_elseif_else)

    def test_0_bug(self):
        """
        bug: does not display 0 when assigned
        """
        self.print_result(MTTests.zero_bug)

    def test_relations_and_arithmetic(self):
        self.print_result(MTTests.relations_arithmetic)

    def test_empty_function(self):
        self.print_result(MTTests.empty_function)

    def test_complicated(self):
        self.print_result(MTTests.complicated)

    def test_external(self):
        self.print_result(MTTests.external)

    def test_return(self):
        self.print_result(MTTests.return_stmt)

    def test_assignment(self):
        self.print_result(MTTests.assignment)

    def print_result(self, prog):
        result = self.parser.parse(prog)
        print
        print result


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

lexing.py

```
import ply.lex as lex

class Mtlex:

    reserved = {'if' : 'IF',
                'then' : 'THEN',
                'else' : 'ELSE',
                'elif' : 'ELIF',
                'def' : 'DEF',
                'for' : 'FOR',
                'while' : 'WHILE',
                'return' : 'RETURN',
                'new' : 'NEW',
                'true' : 'TRUE',
                'false' : 'FALSE'
                }

    tokens = ['NUMBER',
              'PLUS',
              'MINUS',
              'TIMES',
              'DIVIDE',
              'LPAREN',
              'RPAREN',
              'ID',
              'ASSIGN',
              'LCURL',
              'RCURL',
              'STRING',
              'COMMENT',
              'ML_COMMENT',
              'COMMA',
              # 'POINT',
              'DOTOPERATOR',
              'SEMICOLON',
              'COLON',
              'G_OP',
              'L_OP',
              'GE_OP',
              'LE_OP',
              'AND',
              'OR',
              'EQ',
              'NEQ'] + list(reserved.values())

    # Regular expression rules for simple tokens
    t_PLUS    = r'\+'
    t_MINUS   = r'-'
    t_TIMES   = r'\*'
    t_DIVIDE  = r'/'
```

```
    t_LPAREN  = r'\('
    t_RPAREN  = r'\)'
    t_ASSIGN  = r'='
    t_SEMICOLON = r';'
    t_COLON = r':'
    t_LCURL = r'{'
    t_RCURL = r'}'
    t_STRING = r'"(\\.|[^"])*"'
    t_G_OP = r'>'
    t_L_OP = r'<'
    t_GE_OP = r'>='
    t_LE_OP = r'<='
    t_AND = r'&&'
    t_OR = r'\|\|'
    t_EQ = r'=='
    t_NEQ = r'!='
   # t_COMMMENT = r'/\*.*\*/'
    t_COMMA = r','
    t_DOTOPERATOR = r'\.'
    NUMBER =  r'\d+'
    # t_POINT = t_LPAREN + NUMBER + t_COMMA + NUMBER + t_COMMA +
NUMBER + t_RPAREN

    # A regular expression rule with some action code
    def t_NUMBER(self,t):
        r'\d+'
        # BUG: FIX, DON. 0 DOES NOT WORKKKK
        # TODO: UNARY MINUS OR FORCE NEGATIVE NUMBER?
        t.value = int(t.value)
        return t

    def t_ID(self,t):
        r'[a-zA-Z_][a-zA-Z_0-9]*'
        t.type = self.reserved.get(t.value,'ID')    # Check for
reserved words
        return t

    # Define a rule so we can track line NUMBERs
    def t_newline(self,t):
        r'\n+'
        t.lexer.lineno += len(t.value)

    # Skips over comment tokens
    t_ignore_COMMENT = r'\$(.*)(\n)?'

    t_ignore_ML_COMMENT = r'\$\*[^\$\*]*\*\$'

    # A string containing ignored characters (spaces and tabs)
    t_ignore  = ' \t'

    # Error handling rule
    def t_error(self,t):
        print "Illegal character '%s'" % t.value[0]
```

```
        t.lexer.skip(1)

    def build(self,**kwargs):
        self.lexer = lex.lex(module=self, **kwargs)

    def get_lexer(self):
        return self.lexer

    def tok_str(self, data):
        self.lexer.input(data)
        tok_str = ""
        while True:
            tok = self.lexer.token()
            if not tok:
              break
            tok_str += str(tok) + "\n"
        return tok_str

# m = Mtlex()
# data = '''
# map = Flatmap("testmap.dat",500,500) /* hi */
# map.add(block(COBBLE), (0,0,0))
# map.close()
# '''
# m.build()           # Build the lexer
# m.test(data)     # Test it

if __name__ == "__main__":
    m = Mtlex()
    m.build()
    l = m.get_lexer()
    print "Enter a string to be tokenized"
    while 1:
        line = raw_input()
        print m.tok_str(line)
        print "Enter a string to be tokenized”
```

yaccing.py

```
import ply.yacc as yacc
import sys
from lexing import Mtlex
from traverse import *
from preprocess import *

tokens = Mtlex.tokens

precedence = (
    ('left', 'PLUS', 'MINUS'),
    ('left', 'TIMES', 'DIVIDE')
)
```

```
def getyacc():
    return yacc.yacc()

class Node(object):

    def __init__(self, type, children=None, leaf=None, token=None):
         self.type = type
         if children:
              self.children = children
         else:
              self.children = [ ]
         self.leaf = leaf
         self.token = token

    def __str__(self):
        return self.traverse(1)

    def traverse(self, i):
        s = self.type
        indent = "\n" + i*' |'
        if self.leaf != None:
            if isinstance(self.leaf, Node):
                print "Node"
                s += indent + self.leaf.traverse(i+1)
            else:
                s += indent + str(self.leaf)
        for children in self.children:
            s += indent + children.traverse(i+1)
        return s

def p_translation_unit(p):
    '''
    translation_unit : external_declaration
                     | translation_unit external_declaration
    '''
    if len(p) == 2:
        p[0] = Node('translation_unit', [p[1]])
    else:
        p[0] = Node('translation_unit', [p[1], p[2]])

def p_external_declaration(p):
    '''
    external_declaration : function_definition
                         | statement
    '''
    p[0] = Node('external_declaration', [p[1]])

def p_function_definition(p):
    '''
    function_definition : DEF ID LPAREN parameter_list RPAREN LCURL
statement_list RCURL
                        | DEF ID LPAREN parameter_list RPAREN LCURL
```

```
RCURL
    '''
    if len(p) == 9:
        p[0] = Node('function_definition', [p[4], p[7]], p[2])
    else:
        p[0] = Node('function_definition', [p[4]], p[2])

# def p_class_definition(p):
#     '''
#     class_definition : CLASS ID LPAREN ID RPAREN LCURL

# #def p_declaration_list(p):
# #    '''
#    declaration_list : declaration
#                     | declaration_list declaration
#    '''
#
#    if len(p) == 2:
#        p[0] = Node('declaration_list', [p[1]])
#    else:
#        p[0] = Node('declaration_list', [p[1], p[2]])
#
#def p_declaration(p):
#    '''
#    declaration : statement
#    '''
#    p[0] = Node('declaration', [p[1]])

def p_statement(p):
    '''
    statement : compound_statement
              | expression_statement
              | iteration_statement
              | selection_statement
              | class_method_expression
              | function_expression
              | return_statement
    '''
    p[0] = Node('statement', [p[1]])

def p_compound_statement(p):
    '''
    compound_statement : LCURL RCURL
                       | LCURL statement_list RCURL
    '''
    if len(p) == 3:
        p[0] = Node('compound_statement', [], 'emptychange')
    else:
        p[0] = Node('compound_statement', [p[2]])

def p_statement_list(p):
    '''
    statement_list : statement
```

```
                | statement_list statement
    '''
    if len(p) == 2:
        p[0] = Node('statement_list', [p[1]])
    else:
        p[0] = Node('statement_list', [p[1], p[2]])

def p_expression_statement(p):
    '''
    expression_statement : SEMICOLON
                         | expression SEMICOLON
    '''
    if len(p) == 2:
        p[0] = Node('expression_statement', [], 'emptychange')
    else:
        p[0] = Node('expression_statement', [p[1]])


def p_expression(p):
    '''
    expression : assignment_expression
    '''
    p[0] = Node('expression', [p[1]])


def p_assignment_expression(p):
    '''
    assignment_expression : ID ASSIGN NEW initializer
                          | ID ASSIGN assignment_expression
                          | logical_or_expression
                          | function_expression
    '''
    if len(p) == 4:
        p[0] = Node('assignment_expression', [p[3]], p[1])
    elif len(p) == 5:
        p[0] = Node('assignment_expression', [p[4]], p[1])
    else:
        p[0] = Node('assignment_expression', [p[1]])

def p_logical_or_expression(p):
    '''
    logical_or_expression : logical_and_expression
                          | logical_or_expression OR
logical_and_expression
    '''
    if len(p) == 2:
        p[0] = Node('logical_or_expression', [p[1]])
    else:
        p[0] = Node('logical_or_expression', [p[1], p[3]], p[2])

def p_logical_and_expression(p):
    '''
    logical_and_expression : equality_expression
```

```
                            | logical_and_expression AND
equality_expression
    '''
    if len(p) == 2:
        p[0] = Node('logical_and_expression', [p[1]])
    else:
        p[0] = Node('logical_and_expression', [p[1], p[3]], p[2])

def p_equality_expression(p):
    '''
    equality_expression : relational_expression
                        | equality_expression EQ relational_expression
                        | equality_expression NEQ
relational_expression
    '''
    if len(p) == 2:
        p[0] = Node('equality_expression', [p[1]])
    else:
        p[0] = Node('equality_expression', [p[1], p[3]], p[2])

def p_relational_expression(p):
    '''
    relational_expression : additive_expression
                          | relational_expression G_OP
additive_expression
                          | relational_expression L_OP
additive_expression
                          | relational_expression GE_OP
additive_expression
                          | relational_expression LE_OP
additive_expression
    '''
    if len(p) == 2:
        p[0] = Node('relational_expression', [p[1]])
    else:
        p[0] = Node('relational_expression', [p[1], p[3]], p[2])

def p_additive_expression(p):
    '''
    additive_expression : multiplicative_expression
                        | additive_expression PLUS
multiplicative_expression
                        | additive_expression MINUS
multiplicative_expression
    '''
    if len(p) == 2:
        p[0] = Node('additive_expression', [p[1]])
    else:
        p[0] = Node('additive_expression', [p[1], p[3]], p[2])

def p_multiplicative_expression(p):
    '''
    multiplicative_expression : primary_expression
```

```
                                | multiplicative_expression TIMES
primary_expression
                                | multiplicative_expression DIVIDE
primary_expression
    '''
    if len(p) == 2:
        p[0] = Node('multiplicative_expression', [p[1]])
    else:
        p[0] = Node('multiplicative_expression', [p[1], p[3]], p[2])

def p_initializer(p):
    '''
    initializer : ID LPAREN parameter_list RPAREN
                | primary_expression
    '''
    if len(p) == 2:
        p[0] = Node('initializer', [p[1]])
    else:
        p[0] = Node('initializer', [p[3]], p[1])

def p_class_method_expression(p):
    '''
    class_method_expression : ID DOTOPERATOR function_expression
SEMICOLON
    '''
    p[0] = Node('class_method_expression',[p[3]], p[1])

def p_function_expression(p):
    '''
    function_expression : ID LPAREN parameter_list RPAREN
    '''
    if len(p) == 5:
        p[0] = Node('function_expression',[p[3]], p[1])
    else:
        p[0] = Node('function_expression', [], p[1])

def p_parameter_list(p):
    '''
    parameter_list : parameter_declaration
                   | parameter_list COMMA parameter_declaration
                   |
    '''
    if len(p) == 2:
        p[0] = Node('parameter_list', [p[1]])
    elif len(p) == 4:
        p[0] = Node('parameter_list',[p[1], p[3]])
    else:
        p[0] = Node('parameter_list')

def p_parameter_declaration(p):
    '''
    parameter_declaration : initializer
    '''
```

```
    p[0] = Node('parameter_declaration', [p[1]])


def p_primary_expression(p):
    '''
    primary_expression : ID
                       | STRING
                       | NUMBER
                       | TRUE
                       | FALSE
                       | LPAREN expression RPAREN
    '''
    if not isinstance(p[1], basestring) and not isinstance(p[1],int):
        p[0] = Node('primary_expression', [p[1]])
    elif len(p) == 4:
        p[0] = Node('primary_expression', [p[2]])
    else:
        p[0] = Node('primary_expression', [], p[1])

# def p_id_name(p):
#     '''
#     id_name : ID
#     '''
#     p[0] = Node('id_name',[], p[1])

# def p_point_gen(p):
#     '''
#     point_gen : POINT
#     '''
#     p[0] = Node('point_gen',[], p[1])

def p_iteration_statement(p):
    '''
    iteration_statement : WHILE LPAREN expression RPAREN statement
                        | FOR LPAREN expression_statement
expression_statement expression RPAREN statement
    '''
    if p[1] == "while":
        p[0] = Node('iteration_statement', [p[3], p[5]])
    else:
        p[0] = Node('iteration_statement', [p[3], p[4], p[5], p[7]])

def p_selection_statement(p):
    '''
    selection_statement : IF LPAREN expression RPAREN statement
                        | IF LPAREN expression RPAREN statement ELSE
statement
    '''
    if len(p) == 6:
        p[0] = Node('selection_statement', [p[3], p[5]])
    else:
        p[0] = Node('selection_statement', [p[3], p[5], p[7]])
```

```
def p_return_statement(p):
    '''
    return_statement : RETURN SEMICOLON
                     | RETURN expression SEMICOLON
    '''
    if len(p) == 4:
        p[0] = Node('return_statement', [p[2]])
    else:
        p[0] = Node('return_statement')

def p_error(p):
    # we should throw compiler error in this case
    if p == None:
        print "Syntax error at last token."
    else:
        print "Syntax error around line number \n %d : %s " %
(p.lineno, p.value)

data_1 = '''
def makeblocks(start, end, x) {
    while (start < end) {
        c = new Point(0,0,start);
        x.add(block(COBBLESTONE), c);
        start = start + 1;
    }
    for (;start<end;start=start+1)
    {
        c = new Point(0,0,start);
        x.add(block(COBBLESTONE), c);
    }
}

def main() {
x = new Flatmap("testfiles/testmap",500,500,500);
makeblocks(0,10, x);
x.close();
}
'''

data_2 = '''
a = (10,20,30);
'''

data_3 = '''
a = 2
if (a> 1 ) { a = 1;}
'''

data_4 = '''
def foo() {
    return true;
}
def main() {
```

```
    a = 2;
    if (foo()) {
        b = 3;
    }
    x = new Flatmap("testfiles/testmap", 500, 500, 500);
    c = new Point(0, 0, 0);
    x.add(block(STONE), c);
    x.close();
}
'''

# generate the parser
parser = yacc.yacc()
# generate the lexer to be used with parser
m = Mtlex()
m.build()
# preprocessing step
preprocessor = Processor()
data_4 = preprocessor.preprocess(data_4)

result1 = parser.parse(data_4, lexer=m.lexer)
print result1
#
firstline = '''
import logging
import os
import sys
from pymclevel import mclevel
from pymclevel.box import BoundingBox'''
t = Traverse(result1).getpython()
code = firstline + "\n" + t + "\n"
#f = open("hello.py",'w')
#f.write(code)
print code
```

traverse.py

```
import sys
import StringIO
import types
import re

class Traverse(object):

    def __init__(self, tree, file = sys.stdout):
        self.f = file
        self.flist = {"Flatmap": "Flatmap",
                      "block": "materials.blockWithID",
                      "add": "fillBlocks",
                      "close": "saveInPlace"}
        # function argument types for type-checking
```

```
        self.fargs = {"Flatmap": [str, int, int, int],
                      "Point": [int, int, int],
                      "List": "every"}
        self.class_meths = {"LIST": {
                                'append': "every",
                                'get': [int],
                                'delete': [int]
                                }
                            }
        self.class_meth_impls = {"LIST": {
                'append': (lambda name, params : '%s.append(%s)' %
(name, params)),
                'get': (lambda name, params : '%s[%s]' % (name,
params)),
                'delete': (lambda name, params : 'del %s[%s]' % (name,
params))
            }
        }
        # will be used for scope checking
        self.var_scopes = [[]]
        self.scope_depth = 0
        self.flistsymbol = {"close" : "MAP"}
        self.blocks = {"STONE":1,
                       "GRASS":2,
                       "DIRT":3,
                       "COBBLESTONE":4,
                       "WOODENPLANK":5,
                       "SAPLING":6,
                       "BEDROCK":7,
                       "WATER":8,
                       "WATER":9,
                       "LAVA":10,
                       "LAVA":11,
                       "SAND":12,
                       "GRAVEL":13,
                       "GOLDORE":14,
                       "IRONORE":15,
                       "COALORE":16,
                       "WOOD":17,
                       "LEAVES":18,
                       "SPONGE":19,
                       "GLASS":20,
                       "LAPISLAZULIORE":21,
                       "LAPISLAZULIBLOCK":22,
                       "DISPENSER":23,
                       "SANDSTONE":24,
                       "NOTEBLOCK":25,
                       "BED":26,
                       "POWEREDRAIL":27,
                       "DETECTORRAIL":28,
                       "STICKYPISTON":29,
                       "COBWEB":30,
                       "TALLGRASS":31,
```

```
"DEADSHRUB":32,
"PISTON":33,
"PISTON":34,
"WOOL":35,
"PISTON":36,
"DANDELION":37,
"ROSE":38,
"BROWNMUSHROOM":39,
"REDMUSHROOM":40,
"BLOCKOFGOLD":41,
"BLOCKOFIRON":42,
"STONESLAB":43,
"STONESLAB":44,
"BRICK":45,
"TNT":46,
"BOOKCASE":47,
"MOSSSTONE":48,
"OBSIDIAN":49,
"TORCH":50,
"FIRE":51,
"MOBSPAWNER":52,
"WOODENSTAIRS":53,
"CHEST":54,
"REDSTONEWIRE":55,
"DIAMONDORE":56,
"BLOCKOFDIAMOND":57,
"WORKBENCH":58,
"WHEAT":59,
"FARMLAND":60,
"FURNACE":61,
"FURNACE":62,
"SIGN":63,
"WOODDOOR":64,
"LADDER":65,
"RAIL":66,
"COBBLESTONESTAIRS":67,
"SIGN":68,
"LEVER":69,
"STONEPRESSUREPLATE":70,
"IRONDOOR":71,
"WOODENPRESSUREPLATE":72,
"REDSTONEORE":73,
"REDSTONEORE":74,
"REDSTONETORCH":75,
"REDSTONETORCH":76,
"BUTTON":77,
"SNOW":78,
"ICE":79,
"SNOWBLOCK":80,
"CACTUS":81,
"CLAYBLOCK":82,
"SUGARCANE":83,
"JUKEBOX":84,
```

```
"FENCE":85,
"PUMPKIN":86,
"NETHERRACK":87,
"SOULSAND":88,
"GLOWSTONE":89,
"PORTAL":90,
"JACKOLANTERN":91,
"CAKE":92,
"REDSTONEREPEATER":93,
"REDSTONEREPEATER":94,
"LOCKEDCHEST":95,
"TRAPDOOR":96,
"SILVERFISHSTONE":97,
"STONEBRICKS":98,
"BROWNMUSHROOM":99,
"REDMUSHROOM":100,
"IRONBARS":101,
"GLASSPANE":102,
"MELON":103,
"PUMPKINVINE":104,
"MELONVINE":105,
"VINES":106,
"FENCEGATE":107,
"BRICKSTAIRS":108,
"STONEBRICKSTAIRS":109,
"MYCELIUM":110,
"LILYPAD":111,
"NETHERBRICK":112,
"NETHERBRICKFENCE":113,
"NETHERBRICKSTAIRS":114,
"NETHERWART":115,
"ENCHANTMENTTABLE":116,
"BREWINGSTAND":117,
"CAULDRON":118,
"ENDPORTAL":119,
"ENDPORTALFRAME":120,
"ENDSTONE":121,
"DRAGONEGG":122,
"REDSTONELAMP":123,
"REDSTONELAMP":124,
"OAKWOODSLAB":125,
"COCAPLANT":127,
"SANDSTONESTAIRS":128,
"EMERALDORE":129,
"ENDERCHEST":130,
"TRIPWIREHOOK":131,
"TRIPWIRE":132,
"BLOCKOFEMERALD":133,
"WOODENSTAIRS":134,
"WOODENSTAIRS":135,
"WOODENSTAIRS":136,
"COMMANDBLOCK":137,
"BEACON":138,
```

```
                        "COBBLESTONEWALL":139,
                        "FLOWERPOT":140,
                        "CARROT":141,
                        "POTATOES":142,
                        "BUTTON":143,
                        "HEADBLOCK":144,
                        "ANVIL":145,
                        "TRAPPEDCHEST":146,
                        "WEIGHTEDPRESSUREPLATE":147,
                        "WEIGHTEDPRESSUREPLATE":148,
                        "REDSTONECOMPARATOR":149,
                        "REDSTONECOMPARATOR":150,
                        "DAYLIGHTSENSOR":151,
                        "BLOCKOFREDSTONE":152,
                        "NETHERQUARTZORE":153,
                        "HOPPER":154,
                        "QUARTZBLOCK":155,
                        "QUARTZSTAIRS":156,
                        "ACTIVATORRAIL":157,
                        "DROPPER":158,
                        "HAYBALE":170,
                        "CARPET":171,
                        "HARDENEDCLAY":172
                        }
        self.relops = {'<', '>', '<=', '>=', '==', '!=',
                       '+', '-', '*', '/', '%'}
        self.future_imports = []
        self.tempPoints = set()
        # Type table for variables
        self.symbols = {}
        self.values = {}
        self.waitingfor = set()
        self._indent = 0
        self.x = self.dispatch(tree)
        self.f.write("")
        self.f.flush()

    def fill(self, text = ""):
        "Indent a piece of text, according to the current indentation
level"
        s = ""
        buf = StringIO.StringIO(text)
        print "indenting ", self._indent
        for line in buf:
            if self._indent:
                s += "    " + line
            else:
                s += line
        return s

    def getpython(self):
        return self.x
```

```
    def flatten(self, x):
        result = []
        for el in x:
            if hasattr(el, "__iter__") and not isinstance(el,
basestring):
                result.extend(self.flatten(el))
            else:
                result.append(el)
        return result

    def write(self, text):
        "Append a piece of text to the current line."
        self.f.write(text)

    def enter(self):
        "Print ':', and increase the indentation and create a new
scope"
        # initialize depth
        self.scope_depth += 1
        # print "depth ", self._indent
        self.var_scopes.append([])
        self._indent += 1
        return ":"

    def leave(self):
        "Decrease the indentation level and remove out-of-scope
symbols"
        # remove symbols from this scope and then return s
        print self.symbols
        for var in self.var_scopes[self.scope_depth]:
            del self.symbols[var]
            if (var + str(self.scope_depth)) in self.symbols:
                self.symbols[var] = self.symbols[var +
str(self.scope_depth)]
                del self.symbols[var + str(self.scope_depth)]
            if var in self.values:
                del self.values[var]
                if (var + str(self.scope_depth)) in self.values:
                    self.values[var] = self.values[var +
str(self.scope_depth)]
                    del self.values[var + str(self.scope_depth)]
        del self.var_scopes[self.scope_depth]
        self.scope_depth -= 1
        self._indent -= 1
        # print "leaving ", self._indent

    # calls the function corresponding to the name of the node of the
tree. Call on a single node and not a list
    def dispatch(self, tree, flag=None):
        "Dispatcher function, dispatching tree type T to method _T."
        if isinstance(tree, list):
            for t in tree:
                self.dispatch(t,flag)
```

```
            return
        meth = getattr(self, "_"+tree.type)
        x = meth(tree,flag)
        return x

    def _primary_expression(self,tree,flag=None):
        if tree.leaf in self.blocks:
            return str(self.blocks[tree.leaf])
        elif flag == "block":
            raise Exception("Not a valid block type")
        elif tree.leaf == "true":
            return "True"
        elif tree.leaf == "false":
            return "False"
        elif tree.leaf:
            # print str(tree.leaf)
            return str(tree.leaf)
        elif len(tree.children) == 1: # It is a point
            return self.dispatch(tree.children[0],flag)
        else:
            return "0"

    def _class_method_expression(self,tree,flag=None):
        s = tree.leaf
        a = self.dispatch(tree.children[0], s)
        return a

    def _function_expression(self,tree,flag=None): # not complete
        if self.symbols.get(flag) == "MAP":
            if tree.leaf == "add":
                return self.add_method(tree,flag)
            else:
                return flag + "." + self.flist[tree.leaf] + "()"
        elif flag:
            if self.symbols.get(flag) in self.class_meths:
                class_methods =
self.class_meths[self.symbols.get(flag)]
                print tree.leaf
                if tree.leaf in class_methods:
                    params = self.dispatch(tree.children[0],flag)
                    typed_params = [self.num_or_str(param) for param
in params]
                    init_args = [self.get_type(param) for param in
typed_params]
                    if class_methods[tree.leaf] != "every":
                        for (e_p, p) in zip(class_methods[tree.leaf],
init_args):
                            if e_p != "all" and e_p != p:
                                raise Exception("Class Method %s of %s
excepted %s but got %s"
                                    % (tree.leaf, flag,
class_methods[tree.leaf], init_args))
```

```
                    s = self.listtoparams(params)
                    s =
self.class_meth_impls[self.symbols.get(flag)][tree.leaf](flag, s)
                    print s
                    return s
        elif tree.leaf in self.flist:
            if tree.leaf in self.flistsymbol:
                if not self.symbols.get(flag) ==
self.flistsymbol[tree.leaf]:
                    raise Exception(tree.leaf + " method called on a
non " + self.flistsymbol[tree.leaf] + " type")
            return flag + "." + self.flist[tree.leaf] + "()"
        else:
            if tree.leaf not in self.fargs:
                raise Exception("Function %s is not user-defined nor
is it part of the MineTime library"
                    % (tree.leaf))
            if len(tree.children)==1:
                params = self.dispatch(tree.children[0],flag)
                if tree.leaf in self.fargs:
                    typed_params = [self.num_or_str(param) for param
in params]
                    init_args = [self.get_type(param) for param in
typed_params]
                    print tree.leaf, init_args, params, self.symbols
                    if self.fargs[tree.leaf] != "every" and
init_args != self.fargs[tree.leaf]:
                        raise Exception("Function Type Check Error
for %s, expected %s but got %s"
                            % (tree.leaf, str(self.fargs[tree.leaf]),
str(init_args)))
                s = self.listtoparams(params)
                # print s
            else:
                s = ""
            return tree.leaf + "(" + s + ")"

    def add_method(self,tree,flag=None):
        # add must be called on a map type
        if not self.symbols.get(flag) == "MAP":
            raise Exception("Add method called on a non map type")
        a = flag + "." + self.flist[tree.leaf] + "("
        x = self.dispatch(tree.children[0],flag) # x[0] has block with
number, x[1] has point
        if len(x) != 2:
            raise Exception("Wrong number of parameters given to add
method")
        # print "It is:",x[1]
        if not self.symbols.get(x[1]) == "POINT" and not x[1] in
self.tempPoints:
            raise Exception("Not a valid point")
        if x[1] in self.tempPoints:
            self.tempPoints.remove(x[1])
```

```
        p1 = "BoundingBox(origin=" + x[1] + ",size=(1,1,1)),"
        p2 = flag + "." + x[0]
        a+= p1 + p2 + ")"
        return a

    def _expression(self,tree,flag=None):
        return self.dispatch(tree.children[0],flag)

    def _assignment_expression(self, tree,flag=None):
        # print "assignmnet ", tree.leaf
        x = self.dispatch(tree.children[0],flag) # x has name, y has
params
        #print x
        if not tree.leaf:
            return x
        else:
            # add x to the scoping dict to be removed when out of
scope
            self.var_scopes[self.scope_depth].append(tree.leaf)
            # all symbols seen (but may not be defined)
            if type(x) is tuple:
                if x[0] == "Flatmap":
                    self.symbol_add_helper(tree.leaf, "MAP")
                    return self.flatmap_method(tree.leaf, x[1])
                elif x[0] == "Point":
                    self.symbol_add_helper(tree.leaf, "POINT")
                    return self.point_method(tree.leaf, x[1])
                elif x[0] == "List":
                    self.symbol_add_helper(tree.leaf, "LIST")

                    list_init = str([int(e) for e in x[1] if
self.isNum(e)])
                    return "%s = %s" % (tree.leaf, list_init)
            else: # assigning a point right now
                if x in self.tempPoints:
                    self.symbols[tree.leaf] = "POINT"
                    self.tempPoints.remove(x)
                elif self.isNum(x) or x == '0': # int
                    self.symbol_add_helper(tree.leaf, int,
self.isNum(x))
                else:
                    # check if we need to do type conversion
                    relopslist = ['+', '-', '/', '*']
                    if [e for e in relopslist if e in x]:
                        self.symbol_add_helper(tree.leaf,
self.get_inference_type(x))
                    else:
                        self.symbol_add_helper(tree.leaf, float)
                print self.symbols
                return tree.leaf + "=" + x

    def symbol_add_helper(self, var, type_val, value=None):
        if var in self.symbols:
```

```
            self.symbols[var + str(self.scope_depth)] =
self.symbols[var]
            if value and var in self.values:
                self.values[var + str(self.scope_depth)] =
self.values[var]
        self.symbols[var] = type_val
        if value:
            self.values[var] = value

    def listtoparams(self,l,x=None):
        s = ""
        comma = False
        for a in l:
            if comma:
                s += ","
            else:
                comma = True
            s += a
            if x:
                self.waitingfor.add(a)
        return s

    def _logical_or_expression(self,tree,flag=None):
        if tree.leaf:
            s = self.dispatch(tree.children[0],flag) + " or " +
self.dispatch(tree.children[1],flag)
            return s
        return self.dispatch(tree.children[0],flag)

    def _logical_and_expression(self,tree,flag=None):
        if tree.leaf:
            s = self.dispatch(tree.children[0],flag) + " and " +
self.dispatch(tree.children[1],flag)
            return s
        return self.dispatch(tree.children[0],flag)

    def _equality_expression(self,tree,flag=None):
        if tree.leaf:
            s = self.dispatch(tree.children[0],flag) + tree.leaf +
self.dispatch(tree.children[1],flag)
            return s
        return self.dispatch(tree.children[0],flag)

    def _relational_expression(self,tree,flag=None):
        if tree.leaf:
            s = self.dispatch(tree.children[0],flag) + tree.leaf +
self.dispatch(tree.children[1],flag)
            return s
        return self.dispatch(tree.children[0],flag)

    def _additive_expression(self,tree,flag=None):
        if tree.leaf:
            s = self.dispatch(tree.children[0],flag) + tree.leaf +
```

```
self.dispatch(tree.children[1],flag)
            return s
        return self.dispatch(tree.children[0],flag)

    def _multiplicative_expression(self,tree,flag=None):
        if tree.leaf:
            s = self.dispatch(tree.children[0],flag) + tree.leaf +
self.dispatch(tree.children[1],flag)
            return s
        return self.dispatch(tree.children[0],flag)

    def flatmap_method(self, name, param):
        # print "hello" + param[3]
        if self.getint(param[1]) and self.getint(param[2]) and
self.getint(param[3]):
            sizex = self.getint(param[1])
            sizey = self.getint(param[2])
            sizez = self.getint(param[3])
            x = str(int(int(sizex) * 1/2 * -1))
            y = str(0)
            z = str(int(int(sizez) * 1/2 * -1))
            if sizey > 255:
                sizey = 255
            size = "(" + str(sizex) + "," + str(sizey) + "," +
str(sizez) + ")"
            point = "(" + x + "," + y + "," + z + ")"
        else:
            point = "(" + param[1] + "*-0.5," + param[2] + "*-0.5," +
param[3] + "*-0.5)"
            size = "(" + param[1] + "," + param[2] + "," + param[3] +
")"
        fline = "mclevel.MCInfdevOldLevel(" + param[0] + ",
create=True)"
        line = name + ".createChunksInBox(BoundingBox(" + point + ","
+ size + "))"
        comp = name + "=" + fline + "\n" + line
        return comp

    def point_method(self, name, param):
        if len(param) != 3:
            raise Exception("Wrong number of params passed to
Flatmap")
        elif not (self.checkint(param[0]) and self.checkint(param[1])
and self.checkint(param[2])):
            raise Exception("Parameters passed were not integers")
        else:
            self.symbols[name] = "POINT"
            return name + "=(" + param[0] + "," + param[1] + "," +
param[2] + ")"
            #print self.symbols


    def checkint(self,s):
```

```
        if not s : return False
        try:
            ret = int(s)
        except ValueError:
            if s in self.values:
                return self.values[s]
        return True

    def getint(self,s):
        if self.checkint(s):
            try:
                return int(s)
            except ValueError:
                return self.values.get(s)

    def num_or_str(self, x):
        """The argument is a string; convert to a number if possible,
or strip it.
        >>> num_or_str('42')
        42
        >>> num_or_str(' 42x ')
        '42x'
        """
        if hasattr(x, '__int__'): return x
        try:
            return int(x)
        except ValueError:
            try:
                return float(x)
            except ValueError:
                    return str(x).strip()

    def _initializer(self, tree, flag=None):
        if tree.leaf:
            if tree.leaf == "block":
                x = self.flist[tree.leaf]
                y = self.dispatch(tree.children[0],"block")
                return x + "(" + y + ")"
            if tree.leaf in self.flist:
                x = self.flist[tree.leaf]
            else:
                x = tree.leaf
                if x not in self.fargs:
                    raise Exception("Initializer %s not defined in
this language"
                        % x)
            if tree.children:
                params = self.dispatch(tree.children[0],flag)
                # print params
                # initializer argument type checking
                if tree.leaf in self.fargs:
                    typed_params = [self.num_or_str(param) for param
in params]
```

```
                    init_args = [self.get_type(param) for param in
typed_params]
                    if self.fargs[tree.leaf] != "every" and
init_args != self.fargs[tree.leaf]:
                        raise Exception("Initializer Type Check Error
for %s, excepted %s but got %s"
                            % (tree.leaf, str(self.fargs[tree.leaf]),
str(init_args)))
                return (x, params)
            else:
                return x
        else:
            return self.dispatch(tree.children[0],flag)

    def get_type(self, param):
        """given a symbol variable or primary expression, will return
its type"""
        if type(param) == str and not re.search(r'"(\\.|[^"])*"',
param):
            if param in self.symbols:
                return self.symbols[param]
            else:
                raise Exception("Variable %s never initialized within
this scope" % param)
        return type(param)

    def _parameter_list(self, tree, flag=None): # HAVE TO HANDLE
FUNCTION PARAMETERS
        if len(tree.children) == 0:
            return ""
        if len(tree.children) == 1:
            return self.dispatch(tree.children[0],flag)
        else:
            x = self.dispatch(tree.children[0],flag)
            y = self.dispatch(tree.children[1],flag)
            z = [x] + [y]
            return self.flatten(z)
                    # if len(tree.children) == 1:
        #     self.dispatch(tree.children)
        # else:
        #     self.dispatch(tree.children[0])
        #     self.write(",")
        #     self.dispatch(tree.children[1])

    # the following are incomplete and won't work with more
complicated statements
    def _parameter_declaration(self, tree, flag=None):
        return self.dispatch(tree.children[0],flag)

    def _declaration_list(self, tree, flag=None):
        if len(tree.children) == 1:
            return self.dispatch(tree.children[0],flag)
        else:
```

```
            return self.dispatch(tree.children[0],flag) + "\n" +
self.dispatch(tree.children[1],flag)

    def _declaration(self, tree, flag=None):
        return self.dispatch(tree.children[0],flag)

    def _expression_statement(self,tree,flag=None):
        #print "HI",self.dispatch(tree.children[0],flag)
        return self.dispatch(tree.children[0],flag)

    def _statement(self,tree,flag=None):
        return self.dispatch(tree.children[0],flag)

    def _statement_list(self,tree,flag=None):
        if len(tree.children) == 1:
            return self.dispatch(tree.children[0],flag)
        else:
            return self.dispatch(tree.children[0],flag) + "\n" +
self.dispatch(tree.children[1],flag)

    def _expression_statement(self,tree,flag=None):
        if len(tree.children) != 0:
            return self.dispatch(tree.children[0],flag)
        else:
            return ""

    def _compound_statement(self,tree,flag=None):
        if len(tree.children) == 0:
            return ""
        else:
            return self.dispatch(tree.children[0],flag)

    def _point_gen(self,tree,flag=None):
        self.tempPoints.add(tree.leaf)
        return tree.leaf

    def _selection_statement(self,tree,flag=None):
        # print "selection"
        if len(tree.children) == 2: # if statement
            s = "if " + self.dispatch(tree.children[0],flag) + ":\n"
            self.enter()
            r = self.dispatch(tree.children[1],flag) + "\npass"
            # adding the indent yo
            # print self.symbols
            s += self.fill(r)
            self.leave()
            # print self.symbols
            return s
        else:
            s = "if " + self.dispatch(tree.children[0],flag) + ":\n"
            self.enter()
            r = self.dispatch(tree.children[1],flag)
            s += self.fill(r + "\n")
```

```
            self.leave()
            s+= "else:\n"
            self.enter()
            t = self.dispatch(tree.children[2],flag) + "\npass"
            s += self.fill(t)
            self.leave()
            return s

    def _iteration_statement(self,tree,flag=None):
        if len(tree.children) == 2: # while statement
            s = "while " + self.dispatch(tree.children[0],flag) +
":\n"
            # adding the indent yo
            self.enter()
            r = self.dispatch(tree.children[1],flag)
            r = r + "\npass"
            s += self.fill(r)
            self.leave()
            return s
        else: #for statement
            s = self.dispatch(tree.children[0],flag) + "\n" + "while "
+ self.dispatch(tree.children[1],flag)  + ":\n"
            # adding the indent yo
            self.enter()
            r = self.dispatch(tree.children[3],flag) + "\n" +
self.dispatch(tree.children[2],flag) + "\npass"
            s += self.fill(r)
            self.leave()
            return s

    def _external_declaration(self,tree,flag=None):
        return self.dispatch(tree.children[0],flag)

    def _translation_unit(self,tree,flag=None):
        if len(tree.children) == 1:
            return self.dispatch(tree.children[0],flag)
        else:
            s = self.dispatch(tree.children[0],flag)
            t = self.dispatch(tree.children[1],flag)
            return s + "\n\n" + t

    def get_param_types(self, params, tree):
        ''' will return a list of type objects'''
        typed_params = []
        for param in params:
            typed_params.append(self.get_param_type(param, tree))
        return typed_params

    def get_param_type(self, param, tree):
        '''traverse tree until we find spot where param has to be
certain type'''
        if tree.leaf == param:
            if tree.type == "class_method_expression":
```

```
                for class_obj in self.class_meths:
                    if tree.children[0].leaf in
self.class_meths[class_obj]:
                        return class_obj
            else:
                return True
        for child in tree.children:
            ret_val = self.get_param_type(param, child)
            if ret_val:
                if tree.leaf in self.relops:
                    params = self.dispatch(tree.children[0])
                    return int
                if tree.leaf in self.fargs:
                    print "hello"
                    params = self.dispatch(tree.children[0])
                    print params
                    # print "hi " + params
                return ret_val

    def _function_definition(self, tree, flag=None):
        fname = tree.leaf
        s = "def " + tree.leaf + "("
        if len(tree.children) == 2:
            self.enter()
            params = self.dispatch(tree.children[0],flag)

            # find out the necessary types for this new function
            self.fargs[fname] = self.get_param_types(params,
tree.children[1])
            for (param, param_type) in zip(params, self.fargs[fname]):
                print (param, param_type)
                self.symbols[param] = param_type
                self.var_scopes[self.scope_depth].append(param)
            #print self.symbols
            comma = False
            for a in params:
                if comma:
                    s += ","
                else:
                    comma = True
                s += a
                self.waitingfor.add(a)
            s = s + "):\n"
            #print self.waitingfor
            r = self.dispatch(tree.children[1],flag)
            r += "\npass"
            s += self.fill(r)
            self.leave()
        else:
            p = self.dispatch(tree.children[0],flag)
            comma = False
            for a in p:
                if comma:
```

```
                    s += ","
                else:
                    comma = True
                s += a
                self.waitingfor.add(a)
            s = s + "):"+"\n"
            self.enter()
            self.fill("pass")
        return s

    def _return_statement(self, tree, flag=None):
        if tree.children:
            s = "return " + self.dispatch(tree.children[0],flag)
            return s
        return "return "

    def isNum(self, s):
        """Convert string to either int or float."""
        #print s
        #print self.is_same_type(s)
        try:
            ret = int(s)
        except ValueError:
            return False
        return ret

    def get_inference_type(self, s):
        esc_relops = map(re.escape, self.relops)
        delimit = r'|'.join(esc_relops)
        tokens = re.split(delimit, s)

        typ = self.get_type_t(tokens[0])
        for t in tokens:
            if self.get_type_t(t) != typ:
                raise Exception('Type Conversion Error between %s
and %s'
                    % (t, typ))
        return typ

    def get_type_t(self, s):
        try:
            int(s)
            return int
        except ValueError:
            if s in self.symbols:
                return self.symbols[s]
            else:
                # Should not default to str actually. Doesn't exist
should throw
                # error and check for string separately
                return str
```

minetime.py

```
#!/usr/bin/env python

import yaccing as yacc
import sys
from lexing import Mtlex
from traverse import *
from preprocess import *

def main(argv):
    inputfile = argv[1]
    filename = inputfile.split(".")[0]
    source = open(inputfile).read()
# generate the parser
    parser = yacc.getyacc()
# generate the Lexer to be used with parser
    m = Mtlex()
    m.build()
# preprocessing step
    preprocessor = Processor()
    source = preprocessor.preprocess(source)
    tree = parser.parse(source, lexer=m.lexer)

    firstline = '''import logging
import os
import sys
from pymclevel import mclevel
from pymclevel.box import BoundingBox
'''
    lastline = '''
if __name__ == '__main__':
    main()
'''
    result = Traverse(tree).getpython()
    code = firstline + "\n" + result + "\n" + lastline + "\n"
    outputfile = filename + ".py"
    output = open(outputfile, 'w')
    output.write(code)

if __name__ == '__main__':
    main(sys.argv)
```