Gridworld: Final Project Documentation

Andrew Phan, Kevin Weng, Loren Weng, Zikai Lin
Uni: ap3243, kw2538, lw2504, zl2442

December 23, 2015

Professor Edwards
PLT
Columbia University

Contents

1 Introduction

There are many people who play games that also want to start getting into game development. Little do many know that playing a game and developing a game are two completely different things. Making a game involves many game related functions, actions, and rules to keep track of, which not only removes the fun out of creating a game but may also scare the user away from creating a game altogether. The Gridworld language streamlines the game creation experience by offering simple tools to create a node-driven based game. Our goal is to allow the designer to quickly delve into writing Gridworld code with ease, and create a modular game based on the content creator's own storyline, plot, and their own integrated functions.

1.1 Motivation

Since none of us really know how to develop any games, we thought it would be a good idea to make a language that would not only teach us about game design but also let you develop games easily. Our target users are for inexperienced developers, who have little to no experience in programming but still want to make a functional game. For this reason, our language has to be simple. We came up with a node-based language, which does not do any type-checking at compile time. With a node-based implementation, we have nodes that may represent for example a story or an event, which can lead to other nodes. Many games, movies or stories have a linear representation of how plots are developed through time. There is usually a beginning, middle and end, and by using nodes, we think novice programmers can easily understand this concept.

1.2 Project Overview

Gridworld is a simple language used for RPG game design. It gives the node function that can help the user easily develope their own game. Node is something that looks like the function but much easier to understand than the function because it has no argument, function type or return values. Node just look like a small part of the story,so users just need to write many nodes and combine them as an open-end story with setting some choices. The story goes on when the player jumps from one node to another. Also, nodes can be reuse so there is no need for the user to write the same story again in different storyline.

1.3 Gridworld's Focus

- Simplifies game development, requiring little, if any, programming experience.
- Allows the user to design a game world based on their own rules, needs and specifications.
- Offers useful game-related tools and functions for content creator.

1.4 Language Advantages

- Rapid object/class declarations.
- Easy to add and modify attributes.
- Allows for the randomization of objects in the game world.
- Node-based game storyline developed by user via setting choices.

2 Language Tutorial

2.1 Pre-requisites

- Python: required for translation.
- Ocaml/Opam: required for compiling compiler.
- Unix variant/Cygwin with gcc: Used with ocamldep to create Makefile.
- tar: Used to extract .tar file.

2.2 Using Gridworld

Assuming that you have downloaded the "gridworld.tar.gz" file to your user directory, we must untar the file that you just downloaded. First, make sure that you have changed your directory to using "cd ~". There should be a "gridworld.tar.gz" file in your user directory. Finally, type in "tar xvzf gridworld.tar.gz" into the terminal.

Now change to the new source directory with "cd gridworld-src". Type in "make" so that it can create all of the ocaml objects and dependencies. An executable file named "gw" should be ready and fully compiled. We have included an example below on how to compile and run a sample program as well, including other available options as well.

Listing 1: Running a Sample Game

```
cd ~
ls -la
tar xvzf gridworld.tar.gz
cd gridworld-src
make
./gw <samplegames/textAdventure.gw > output.py 2>&1  # or replace textAdventure with another name such as pokeySim.gw or smallTestgame.gw
python output.py
```

Listing 2: Other Options

```
make clean # will remove object files, log files, executables etc
make test # will start testing individual components of the compiler
```

3 Language Manual

3.1 Lexical Conventions

3.1.1 Tokens

3.1.1.1 Keywords

Keywords are pre-defined words in the Gridworld compiler. Each keyword has has their own function. The keywords are reserved in the compiler and therefore cannot be used as variable names. Below is a table of reserved keywords in Gridworld:

Keywords	
if	else
while	int
bool	str
node	print
goto	list
choose	readint
readstr	function
return	main
roll	

3.1.1.2 Identifiers

An identifier is an element in the Gridworld language that has been given a name for a particular function, and variables. An identifier begins with a letter or underscore followed by any alphanumeric characters. Special characters such as punctuation and brackets are not allowed, except for the underscore. Identifiers should give a clear indication of what the label does, if it is an action then a name should define that action. Finally, it should not be a keyword.

Listing 3:

```
string player1_Name // player1_Name is the
    identifier
int hitPoints // hitPoints is the identifier
int sum     // sum is the identifier
bool attackPlayer        // attackPlayer is the
    identifier
```

3.1.2 Whitespaces

Whitespaces is any character or a series of whitespace characters that are unused or space between objects. Their purpose is to separate tokens and format programs. Whitespace characters are usually typed in by using the return, spacebar or the tab key. Gridworld ignores all whitespaces, as the language uses the spacing for differentiating tokens. If we look above at code listing ?? above, int and sum are separated by a whitespace. Unlike Ocaml, Gridworld does not care about indentations.

3.1.3 Comments

Like most programming languages, Gridworld supports single line and also multi-line commenting. The language encourages readable, organized and logical source code. With that in mind, being able to comment source code not only helps debug Gridworld, but also helps clarify what a function does, why the creator decided to do something, or to create notes and reminders on what the next step is in terms of game development.

Symbol Syntax	Uses	Example
/* */	Multi-line comments, nested comments	/* This is a multi-line comment, intended to allow the user to provide more info on what's happening during execution */

3.2 Types/Meaning of Identifiers

3.2.1 Primitive Types

The Gridworld languages uses several primitive types in order to describe the user's game environment. The primitive types are listed below and may be used, although not limited to, for example describing a world with hitpoints, weapon damage, name, and dialog of players, monsters and other objects.

Primitive Type	Description	Range	Example
Int	A 32 bit Integer type	-2.147.483.648 to 2.147.483.647	int maxLife = 2000;
Bool	A Boolean value	True (Binary 1) / False (Binary 0)	bool playerGodlike = TRUE;
String	A sequence of characters	Not Applicable	String bossName = ”Diablo”

3.3 Expressions and Operators

The precedence of expression operators are indicated by the order of the following subsections, from highest precedence to the lowest.

3.3.1 Primary Expressions

Identifiers: An identifier refers to a variable or a function.

Constants: A constant can be a number, string, boolean etc. with the different types defined in lexical conventions.

String literals: String literals are directly translated to strings by the compiler.

Parenthesized expressions: The expression is equivalent to the result without parentheses, but the presence of parentheses indicates the precedence as a primary expression.

3.3.2 Unary Operators

3.3.2.1 Logical Negation

Types used with the logical negation operator are Bool and Int. The result of the logical negation of a Bool is true if the value of the expression

is false, and false if value is true. The result of the logical negation of an Int is 1 if the value of the expression is 0, and 0 if the value of the expression is non-zero.
!expression

3.3.3 Function Call

To call a function, it must have been declared and defined previously. A function call has the form functionName(expression1, expression2,...), following the form defined in its declaration. The result is a value of the type defined as a return type in the function declaration.

3.3.4 Multiplicative Operators

The multiplicative operators are left associative. Types of both expressions used with the * operator are Int. The result is the first expression multiplied by the second.
$\boldsymbol{expression * expression};$

Types of both expressions used with the / operator are Int. The result is the first expression divided by the second, and division by zero is not allowed.
$\boldsymbol{expression\ /\ expression};$

Types of both expressions used with the % operator are Int. The result is the remainder from the division of the first expression by the second. Division by zero is not allowed.
$\boldsymbol{expression\ \%\ expression};$

3.3.5 Additive Operators

The Additive operators are left associative. Types of both expressions used with the + operator are Int.

The result is the sum of the two expressions.
$\boldsymbol{expression + expression};$

Types of both expressions used with the - operator are Int. The result is the first expression minus the second.
$\boldsymbol{expression - expression}$;

3.3.6 Relational Operators

The relational operators are left associative. The type of the relational operators are Int.

The result is of type Bool and the value is true if the first expression is less than the second expression, and false otherwise.
expression < expression;

The result is of type Bool and the value is true if the first expression is greater than the second expression, and false otherwise.
expression > expression;

The result is of type Bool and the value is true if the first expression is less than or equal to the second expression, and false otherwise.
$\boldsymbol{expression <= expression}$;

The result is of type Bool and the value is true if the first expression is greater than or equal to the second expression, and false otherwise.
$\boldsymbol{expression >= expression}$;

3.3.7 Equality Operators

The equality operators are left associative. Types used with equality operators are Int, Bool, and String.

The result is type Bool and the value is true if both expressions have the same value, and false otherwise.
$\boldsymbol{expression == expression}$

Types used with equality operators are Int, Bool, and String. The result is type Bool and the value is false if both expressions have the same value, and false otherwise.
expression* != *expression

3.3.8 Boolean Operators

The boolean operators are left associative. Types used with the Boolean operators are Bool.

The result is type Bool and the value is true if both expressions are true and false otherwise.
expression & expression

The result is type Bool and the value is true if at least one of the expressions is true and false otherwise.
expression | expression

3.3.9 Assignment Operators

The assignment operators are right associative. The types used with the assignment operators are Int, Bool, String. Assignment stores the value of the second expression in the first expression, both expressions must be of the same type.
expression* = *expression

3.4 Declarations

3.4.1 Type Specifier

The type specifiers are int, bool, and string.

3.4.2 Variable Declaration

The variables can be initialized with a constant, literal value, or an expression as long as the type of the value and the type of the variable are the same. Variables are declared as:
typeSpecifier varName

3.4.3 Node Declaration

Nodes consist of a node header and a node body. The header takes the form of: **node nodeName{}**

In the example above, the node body would be enclosed in the brackets after the node header.

3.4.4 Function Declaration

Functions consist of a function header. The header takes the form of:
typeSpecifier function fname (params)

3.5 Statements

3.5.1 Expression Statement

Expression statements are the most common form of statement, which are simply of the form:
expression;

3.5.2 Conditional Statement

There are two basic forms of conditional statements:

1. **if (expression) statement**

2. **if (expression) statement else statement**

The expressions must be of type Bool and if the value is true, the statement directly after it will be executed, and once one is executed, any expressions afterwards will not be considered.

3.5.3 Loop Statement

The while statement has the form:
while (expression) statement

The statement is executed repeatedly as long as the value of the expression remains true. This expression is checked before each execution.

3.5.4 Return Statement

A function returns to its caller by the return statement. If an expression follows return, the value is given to the caller of the function and must be of the type specified by the function.
return expression;

3.5.5 Built-in Function Statement

All the built-in functions are parts of the statements, to call the built-in functions like the functions you defined.
print("Hello World");

3.6 Built-in Functions

3.6.1 print

The print function outputs text to either stdout or a file. The first parameter, being the output text, must be a string. The second parameter, also a string, specifies the filepath of output. The absence of the second parameter causes print to default to stdout.

Listing 4: Print

```
// print example
print("Hello gridworld!");
```

3.6.2 readInt()

The readInt function stores the user integer input into the specified variable.

Listing 5: readInt()

```
// readInt() example
health = "100";
readInt(health);
```

3.6.3 readStr()

The readStr function stores the user String input into the specified variable.

Listing 6: readstr()

```
// readStr() example
defaultNPC_text = "Hello Traveler!"
readStr(defaultNPC_text);
```

3.6.4 list()

The list function takes any number of string inputs and prints them in a numbered list

Listing 7: List

```
// list() example
list("go up","go down","go home")
```

3.6.5 choose

The choose function takes any number of nodes as input and prompts the user for an integer input that moves them to said node.

Listing 8: Choose

```
// choose example
list("go up","go down","go home")
choose(sky,underground,home)
```

3.6.6 goto

The goto function takes a single node as an input and moves the user to said node.

Listing 9: Goto

```
// goto example
print("you fall into a hole")
goto(hole)
```

3.6.7 roll

Roll generates a random integer ranging from 1 to 6. This simulates the rolling of a 6-sided die.

Listing 10: roll

```
// roll example
int dice = 0;
roll(dice)
```

3.7 Language Scope

3.7.1 Global Scope

All variables defined at the top-level of a program will be by default part of the global scope, and be visible, modifiable to the entire program. Be careful, to define the variables at the beginning of the program. If a variable is defined at the middle of the program, code written before this variable declaration can not access to this variable.

Listing 11: Global Scope

```
int sum;
int moverange;
bool canMove (int a, int b){
    sum = a + b;//able to use varibale sum
    abs_distance = abs(a - b);//unable to use variable abs_distance
    if (abs_distance <= moverange) return true;
    else return false;
}
int abs_distance
```

3.7.2 Scope within a Function/Node

Variables defined within the function and the mode are the local variables to that function, and it will be expired automatically when the function ends. Codes outside that function can not access or modify those variables.

Listing 12: Scope with a Function

```
int function sum (int a, int b){
    return a+b;
}
a = a - b;//unable to use a, b outside the function sum()
```

Listing 13: Scope with a Node

```
node sum (int a, int b){
    c= a + b;
    return c;
}
c = 10;//unable to use c outside the node sum()
```

3.8 Sample Game Code

3.8.1 Pokeyman Sample Game Code

Listing 14: Pokeyman Sample Game

```
object player
object pokeyman
#can we use print and println?
node start{
    print ("Welcome to Pokeymans, the premium creature slaying game in the world!")
    trainer = new player()
    print ("What's your name?")
    trainer.name = read();
    trainer.gold = 100;
    trainer.balls = 0;
    print ("Are you a boy or a girl?")
    list("Boy","Girl")
    choice = read();
    if(choice ==1){
        trainer.gender = "boy"
    }
    elif(choice ==2){
        trainer.gender = "girl"
    }
    else{
        trainer.gender = "undefined"
    }
    print("Choose your starter pokeyman!")
    list("Charitard","Blakbois","Penusaur")
    choice = read();
    if(choice ==1){
        tempPokey = new pokeyman;
        tempPokey.name = "Charitard"
        tempPokey.lvl = 1
        addSkills(tempPokey,"Hot Breath","Spicy Breath")
        trainer.starter = tempPokey
    }
```

Listing 15: Pokeyman Sample Game

```
   elif(choice ==2){
        tempPokey = new pokeyman;
        tempPokey.name = "Blakbois"
        tempPokey.lvl = 1
        addSkills(tempPokey,"Splash","Waterboard")
        trainer.starter = tempPokey
    }
    elif(choice ==3){
        tempPokey = new pokeyman;
        tempPokey.name = "Penusaur"
        tempPokey.lvl = 1
        addSkills(tempPokey,"Thorn Whip","Throw Leaf")
        trainer.starter = tempPokey
    }
    else{
        tempPokey = new pokeyman;
        tempPokey.name = "Potatochu"
        tempPokey.lvl = 1
        addSkills(tempPokey,"Hurp","Derp")
        trainer.starter = tempPokey
    }
    print("Time to start your adventure!:")
    goto(mainMenu)
}
node mainMenu{
    while (1){}
        print ("What would you like to do?")
        list("Battle!","Shop!","Procrastinate!")
        choose(initBattle,shop,procrastinate)
    }
}
```

Listing 16: Pokeyman Sample Game

```
node procrastinate{
    #IMPORTANT: FIGURE OUT FORMATTING FOR FOR LOOPS
    sum = 0
    for(i from 1 to 2,1){
        sum = sum + roll(6)
    }
    print("you roll 2 dice and you get a total of ")
    print(sum)
    goto(mainMenu)
}
node shop{
    print ("you are in the store")
    print ("you have :")
    print(trainer.gold)
    list("buy a sandwich (20g)","Buy a pokeyball (10g)","Leave")
    choose(sandwich,buyBall,leaveShop)
}
node sandwich{
    rand = roll(3)
    if(trainer.gold>=20){
        trainer.gold = trainer.gold-20;
        if(rand ==0){
            print("you ate a molded sandwich... Disgusting! (HP-10)"
            trainer.hp = trainer.hp - 10
        }
        else{
            print("you ate an overpriced sandwich... The inflation these days... (HP+10)"
            trainer.hp = trainer.hp + 10
        }
    }
    else{
        print("you can't afford it..."
    }
    goto(shop)
}
```

Listing 17: Pokeyman Sample Game

```

node buyBall{
    if(trainer.gold>=10){
        print ("you buy a designer pokeyball(TM)"
            )
        trainer.gold-=10;
        trainer.balls = trainer.balls+1
    }
    else{
        print("you can't afford it..."
    }
    goto(shop)
}
node leaveShop{
    goto(mainMenu)
}
node initBattle{
    enemy = new pokeyman()
    enemy.name = "ratatatatata"
    enemy.lvl = trainer.starter.lvl
    addSkills(enemy,"bite","rabies")
    goto(fight)
}
node fight{
    print("you are fighting a")
    print(enemy.name)
    print("what will you do?")
    list(trainer.starter.skill[0],trainer.starter
       .skill[1],"use pokeyball","flee")
    choose(useSkill(trainer.starter.skill[0],
       enemy),useSkill(trainer.starter.skill[1],
       enemy),useBall,flee)
    useSkill(enemy.skill[0],trainer)
    goto(processBattle)
}
```

Listing 18: Pokeyman Sample Game

```

node processBattle{
    if (enemy.hp<=0){
        goto(victory)
    }
    elif (trainer.hp<=0){
        goto(defeat)
    }
    else{
        goto(fight)
    }
}
node useSkill(skillName,target){
    print(target.name)
    print(" takes ")
    if(skillName = "Hot Breath"){
        print("10");
        target.hp-=10
    }
    if(skillName = "Spicy Breath"){
        target.hp-=10
    }
    if(skillName = "bite"){
        target.hp -=10
    }
    print(" damage")
}
```

Listing 19: Pokeyman Sample Game

```
node useBall{
    if(trainer.balls >=1){
        print("you throw the ball as hard as you
           can and deal critical damage to the
           enemy")
        trainer.balls = trainer.balls-1
        enemy.hp = 0
    }
    else{
        print("you don't have any balls you dunce
           ")
    }
    goto(fight)
}
node flee{
    print("only cowards flee, you lose the game")
    goto(end)
}
node victory{
    player.hp = player.hp + 30
    print("VICTORY")
    gold = roll(100);
    player.gold = player.gold + gold
    goto(mainMenu)
}
node defeat{
    print("You have been defeated. That was a
       shameful display. Go home.")
    goto(end)
}
main{
    goto(start)
}
```

3.8.2 Small Test Game

Listing 20: Small Test Game

```
int gold = 0;
string name = "";
main{
    goto(start);
}
node start{
    print ("welcome to the game!");
    print ("enter your name:");
    readName = "";
    readStr(readName);
    name = readName;
    print("\ngreetings");
    print(readName);
    print("\nHow much gold do you want?");
    readInt(gold);
    list("go to the store","go home");
    choose(store,lose);
}
node store{
    print ("you are in the store");
    print ("you have: ");
    print(gold);
    print("gold");
    list("buy a sandwich (20g)","win (0g)");
    choose(sandwich,win);
}

node sandwich{
    if (gold>20){
        print("you bought a sandwich");
        gold = gold-20;
    }
    else{
        print("you can't afford a sandwich...");
    }
    goto(store);
}
```

Listing 21: Small Test Game

```
node win{
    print("you win!\nCongratulations!");
    print(name);
}
node lose{
    print ("Who goes home as a first choice? You
       lose");
}
```

4 Project Plan

4.1 Stage of Prototype Development

A large scale project needs many steps. Planning, specification, development and testing are the 4 steps to complete our project.

During the planning, we met and talked about topics about the previous projects and list them as choices. Then we discussed about how we were familiar with these topics, and whether we had enthusiasm to work on this topic in a whole semester. When it came to the game topic, we both had interests to work a project about it, and that's the process we decided the topic. After that we decide the time to meet every week if we can.

Once we decided our project topic, we needed to specified it for the future working. We had another meeting to talk about the game feature in our project. We decided to do a board game so we called our project Gridworld(even now it is no longer a board game maker). We designed the board game maker with the board design, charter design and object design. Combining these three features and some built-in functions, such as save, load, undo and so on, it should be an excellent board game maker.

Learning Ocaml was difficult and took some time to learn. Having the scanner, parser, AST, code generator and other files took longer to develop than anticipated. We began our language from the "Hello World", which is the basic function for a language. After that, we add the calculation, variable assignment and access, judgement and loop statements into the language to implement the gcd() function. With the basic components of a language implemented, we tried to add some game features into the language to fulfill its functions and check if the new function worked well. By adding new features one by one, we finally get our language.

Testing is needed during the coding stage or optimization stage after the project is done. We referenced the regression test in the microc, and developed the regression test file for our own. By looking at the LRM, we added enough tests for each tokens, functions and logical operations to make sure our language was robust during the development stage.

4.2 Style Guide

We used the following rules when writing our code to ensure maximum readability:

- Each line of code should remain under 100 characters
- Write utility functions for commonly reused code
- Use camelcase function names and lowercase type names

4.3 Project Timeline

Time	Event
Sept 8 - Sept 15	Project team formed.
Sept 16 - Sept 30	Topic selected and proposal submitted.
Oct 16	Meet with TA and receive proposal feedback.
Oct 18 - Oct 26	LRM done and submitted, parser, scanner, AST started.
Oct 27 - Nov 4	Meet with TA, LRM feedback returned.
Nov 5 - Nov 15	Parser, Scanner, AST, Pretty printer done with basic features.
Nov 16	Hello World and gcd demo to professor. Linear Test Reg. incomplete.
Nov 17	Meeting with TA. Demo hello world and ask why test does not work.
Nov 20	Meeting with TA. Talk about our game and what to do.
Nov 22 - Nov 30	More features added into the language, test script started
Dec 2	Meeting with TA
Dec 3 - Dec 10	Regression test script and some test cases done
Dec 10 - Dec 13	Worked on SAST and Semantic analyzer
Dec 14 - Dec 17	More tests cases added and optimized the language
Dec 17 - Dec 19	Worked on the Final Report
Dec 19 - Dec 20	Prepared for presentation to professor
Dec 21	Presentation and final demo to professor
Dec 22	Proofread then Submit Final report and Code

4.4 Roles and Responsibilities

4.4.1 Roles

Student	Roles
Andrew Phan	Project Manager/Latex Document Organizer/Testing
Loren Weng	Language Developer
Kevin Weng	Systems Guru
Zikai Lin	Testing and Validation

4.4.2 Responsibilities

Student	Responsibilities
Andrew Phan	Email TA/Prof, Doc. submission, Overleaf, Trello, Github, UML diagram
	Meetings and Deadlines, GoogleDocs, Soft. Deployment, Powerpoint
	Final Document, Proofreading, Makefile, Testing, RegTest script, RefManual
Loren Weng	Game Features and Design, AST, Parser, Compiler writing.
Kevin Weng	Language structure design. AST, SAST, Parser, Scanner, Analyzer writing.
Zikai Lin	Test Cases, Regression Test Script, Powerpoint.
	Report what works and what does not. Document writing.

As a group, we have noticed that each member has different strengths and weaknesses. We have therefore assigned each individual in the group with their own responsibilities. As a collective group, we did not strictly impose limitations. If a team member does not have any work in the pipeline then they will be assigned to work on something else. Each individual team member has more or less played an integral part in every phase of the development process, meaning that everyone had to work on the latex pdf document and also delve deep down into ocaml code for testing and validation. If a member was stuck on a problem, it was common for another member to help and solve the problem together. We have noticed that peer programming makes programming a lot less stressful and that GitHub is a great middleman, which allows everyone to share and improve their code.

4.5 Software Development Environment

Each member of the group used a variety of tools. The only mandatory software was Github and Ocaml. Half of our group used the Windows operating system but instead of installing Cygwin, it seemed easier to install VMware Workstation 12 and just virtualize a linux distribution such as Ubuntu. From there, we could setup Github to track and manage source code changes, source code using sublime text and compiling with ocaml, without having to worry about whitespaces in directory structures or other incompatibility issues between operating systems.

Testing was done with the linear regression tester, which we had to rewrite from the microc testall.sh file provided by Professor Edwards. We also used menhir to see if there were any problems with parser.mly, which proved to be quite useful because it generated the automaton and displayed the shift/reduce conflicts if there were any. This was installed via the ocaml package manager called Opam.

In terms of organization and work management, we used a mixture of Google Docs, Overleaf, and Trello. Initially, Google Docs was used to organize everything and also allow us all to simultaneously edit one document, with all of the pertinent information and deadlines. But this was later changed to accomodate the two programs Overleaf and Trello. Overleaf does the same thing as Google Docs, however, it does it better when it comes to .pdf files mostly because it supports the popular Latex language. It was more convenient for everyone to edit the latex file at the same time instead of having just one person recompiling the .tex file each time there is a change to the entire document. Trello was used to share deadlines, important links, to-do lists and attachments. It would send notifications to each member's email when there are any changes to the project board.

4.6 Project Log

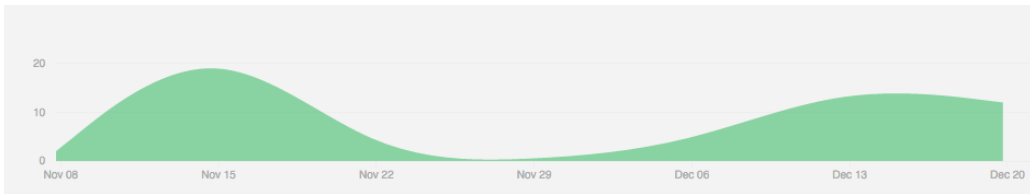

Figure 1: GitHub Commit Analysis

Github has provided us with an idea of how many commits we have made over the course of our project. We have also included our "git log", which shows the date, time and comments made during our commits. The log can be found under the Appendix section or more specifically in Gridworld Project Log. Please keep in mind that some users are not represented as accurately as some of my team members decided to use git for cloning the repository only.

5 Architectural Design

5.1 The Components of our Translator

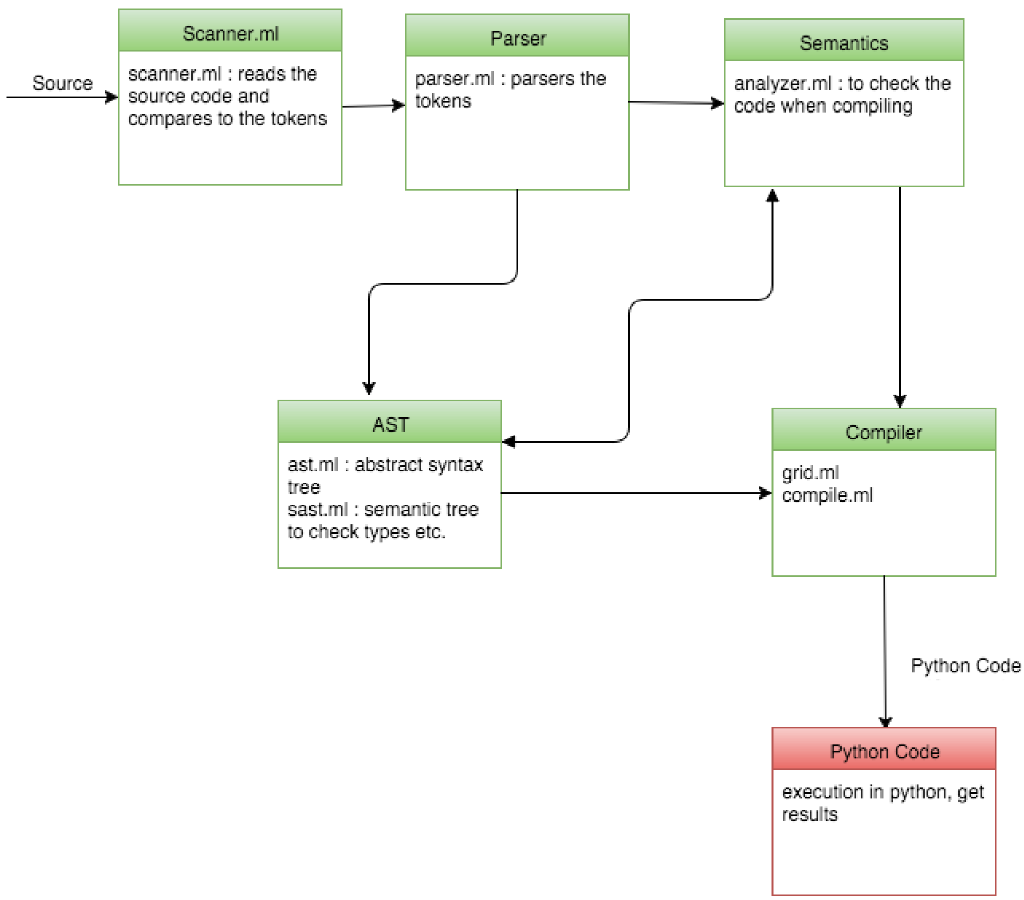

Figure 2: UML Block diagram of Major Translator Components

5.2 Interfaces between Components

The gridworld compiler has the following components:
front-end : scanner, parser, AST.
back-end : SAST, analyzer, and compiler

The block diagram above shows a brief overview of each component, and in this part, their interfaces between each other are described more clearly.

The scanner of a compiler, as its name shows, will scan through the stream of characters from the source code and change them to the recognizable tokens for the parser. It will recognize the commented parts and the whitespaces for the program style and remove them. And since our compiler translates the source code into Python code, the scanner will also recognize some meaningful spaces for the indent, which is necessary in the Python code. Finally, other useful characters will be transferred into the tokens.

The parser sequence the tokens from the scanner into an abstract syntax tree(AST) with the help of the ast.ml. It analyzes the sequence of the tokens and produces the structure that the Gridworld language has. Here, the parser will checks the syntax and catch the syntax errors, but it will not check for type or semantics, which will be done by the semantic analysis part (SAST).

The analyzer will check the type and semantics for every line of code. In semantically-checked Abstract Syntax Tree (SAST), it gives the exact types and helps to keep track of the types when the python code generates.

After the semantics check, the compiler is able to generate Python code by matching the declarations, functions, built-in functions in the main. The compiler should able to distinguish the functions users defined and the built-in function, and if users define the same function name as the default one, it should not able to pass the compiling.

Finally, the Python code is produced and is now ready to be executed. For one of our sample games, we have saved the user from writing at least 200 lines of code.

5.3 Component Implementation/Responsibilities

Student	Components
Andrew Phan	Makefile, grid.ml
Loren Weng + Kevin Weng	Scanner, parser, AST, SAST, analyzer, compile.ml
Zikai Lin + Andrew Phan	Scanner, AST, parser in the beginning.

6 Testing Plan

6.1 Source Language Programs and Target Language Program

Listing 22: example1: Source

```
main{
    print(5 + 2 + 1);
    print("hello world");
}
```

Listing 23: example1: Target

```
print 5 + 2 + 1
print "hello world"
```

Listing 24: example2: Source

```
main{
    int a = 5;
    if (a <= 5){
        a = a + 5;
        print(a);
    }
}
```

Listing 25: example2: Target

```
a=5
if a <= 5:
        a = a + 5
        print a
```

Listing 26: example3: Source

```
main{
    int a = 0;
    while(a < 5){
        print(a);
        a = a + 1;
    }
}
```

Listing 27: example3: Target

```
a=0
while (a < 5):
        print a
        a = a + 1
```

6.2 Test Suites Used

6.2.1 Token and Logic

We created the automated tests for the parsing, scanning, and translation of the AST. All of the tests are chosen based on the tokens in the AST and provides some basic logical calculations.

6.2.2 Game

We created the game by using our language to determine whether the game is running as intended. Since we wanted some level of user interaction, we could not write an automated test script because in our sample games we prompt the user who plays the game to type in a number. The output is therefore based on what the user types. For this reason, we decided not to write an automated test script for these sample games. Instead, we tested it manually to see if the output is the same as our expectations.

6.3 Test Cases and Why we chose them

We tested our language from the following parts: the basic calculations, the logical operation, variables, functions, and built-in functions. In the basic calculations, we tested the binary operators one by one, which also includes the logical operators. Secondly, we tested the precedence of the these operators and the calculations of different value types. We tested the assignments, access to the variable, some calculations of the variables and the scope of the variables. Furthermore, we tested the declarations and calls of the functions, using different types of arguments. Finally, we tested our built-in functions and made sure it can be called correctly. We consider all aspects of our project and gave enough tests(actually we reference the tests in microc). In the test, we tested each part separately case by case so that when something failed, we would immediately know what the problem was.

6.4 Type of Automation used in Testing

Testing each part of the project manually is tedious and requires more work than necessary so we took the professor's advice to use a linear regression test, similarly to his microc testall script. Regression tests (having a shell script to run the small tests automatically, compares the results, and gives the outcome of whether a component of our compiler is working or not) help us test our project efficiently, especially after we had made some changes to our code. Every change in the project may cause some unexpected changes in the language so we should perform a linear test regression every time there has been any new changes to our code. Thus a regression test can quickly check every little component of our compiler to see if anything fails.

Automation test is done by using a shell script (the code is shown after). In the script, it will run all of the test cases in the ./tests directory and compare the results executed to the .out file of the same base name. It will display an "OK" if the result is the same as the expected output otherwise it will show "FAILED" if the result is different and prints the difference in the .diff file, which deals with the validation part of this process. The developer can then quickly know what the problem is and can then modify the code accordingly. The linear regression test is another useful way of debugging our compiler.

7 Conclusion

7.1 Lessons Learned

7.1.1 Andrew Phan

I have learned that it is never a good idea to start late on any project. This especially goes for programming because something always comes up. Maybe you don't have the necessary tools to do the work you set out to do or because more bugs were introduced after you git pushed something to a repository. Either way, you have to spend additional time on the logistics of the entire process. Making decisions and communicating with each other is extremely difficult and is not something to be taken lightly. I would almost equate project management to a full-time job that nobody wants to do. It is a necessary evil, unfortunately. Someone has to do it. I felt bad for having to remind my team members about deadlines and what they had to do but on the other hand I needed to get things done because PLT was not my only commitment.

On top of that, there is quality control, something that not many people account for until the very end. Proofreading and checking whether or not something works can only be done when you presumably have something tangible and presentable. Thus it is necessary to map out everything and finish things at least on time or before the deadline. Project management is not just about managing people and the work they have to do but it encompasses everything that pertains to the project. I also learned the TA is there to help you and that even if the entire team could not make it, you should still meet up and talk about the project.

I had my doubts about creating a game language because I do not have much experience with developing games. I have not seen many languages that focuses on games either. Also, our group does not entirely consist of Computer Science majors, which is another reason why I was a little hesitant on making a language for games. The group as a majority was very nonchalant about almost every decision and frankly I would have appreciated some more enthusiasm. I learned that a project is only as good as the effort you put into it and that depending on other people is going to be a part of working at a larger establishment whether you like it or not. Choosing a project that everyone in the group wants to do is better than doing something just for the sake of doing it, which is why I think transfer students and those who do not know many computer science majors are at a slight disadvantage.

I learned about Trello in my User Interface Design class this semester. It seemed to have a linear way of allowing multiple users to track the progress of a project and I thought it could be of use for this PLT project. The reason for introducing it was because I could easily distribute links and attachments to whoever was invited to the Trello board and I could also add deadlines similar to an online calendar.

Although I have gone over some similar concepts (NFA, DFA) in Computer Science Theory, this class was a lot harder due to the functional programming language Ocaml. After spending a little time with it, it is one of those love hate relationships because you appreciate what it can do but figuring it out is a long and painful process. I learned how useful it is to integrate Merlin with vim, menhir to check for shift/reduce errors and also ocamldebug to step through a program to find out what is wrong with it.

7.1.2 Kevin Weng

There were a number of things that I learned from this project, especially as my first large programming assignment to be done as a group. There were difficulties for each person to complete their assigned task as it took us a while to get the end to end complete compiler working. This meant that while were theoretically finished with their part, they were unable to test it as other parts of the code weren't finished yet. This likely applies to most programming projects, not just for a compiler and we should probably have put more emphasis on getting the bare-bones compiler done before adding more complex parts about the language. However, once this was done, GitHub allowed for easy version control and for each person to add to certain parts of the code which made the programming go much smoother. Another thing I learned was just how important the initial design of the language was. We made many changes to the design of our language as we realized that certain parts wouldn't work very well or we needed more functions to be implemented; this made writing the code very inefficient, as often, after a change, entire parts of the code had to be scrapped and rewritten. Giving more thought and having a more complete idea of the exact design of our code would likely have made things easier.

7.1.3 Loren Weng

What I took away from this project was a deeper understanding of compilers, as was probably the point of the assignment. It's fine to talk about how compilers function theoretically and draw an abstract syntax tree, but having to go knee deep in writing your own definitely is a different task altogether. Throw in other group members working at the same time and we have a party. The importance of version control has been imprinted onto me. The most important thing I've learned from this experience is that sometimes you just have to reduce some of your innate civility to get a group working. People need to admit when they don't know or can't do something, so that work can be redistributed to people that can do it. I honestly feel that the group formation process in the class could be improved, as arbitrarily creating a group out of people that later figure out what they want seems less productive than people spearheading a project and having other people join the group.

7.1.4 Zikai Lin

I regret having chosen this class without any prior computer science knowledge. I didn't know what a compiler was before this lecture and it took me a lot of time to understand this concept. But, understanding one thing is different from working that thing out by yourself, the latter is much more harder. Although we decided our project's content quickly, we wasted many hours on our project due to the fact that we scheduled to meet weekly but plans fell through due to group members, who did not have the time to meetup until a couple of days before a deadline. Meeting regularly is important especially for a project of this size.

The complaints stop here. Now, let me talk about some positive parts of this project. To work as a group, I learn many platforms. For example, use Overleafs to write the proposal and LRM together, use Github to work on the same code, use Trello to manager our work schedule. Both of these are very helpful and can be used in the future. Also, applying the knowledge to something more hands on is important.

We learned a lot in the lectures and from doing the homework assignments. However, when I actually worked on the program, I find that I maybe did not learn as much as I wanted before starting this project. When shift/reduce conflicts were found in the code, I fixed it and learned much more about these concepts compared to the homework assignments. Also, I was responsible for the testing part of our project and learned a lot when nothing worked. Before this project, I have no idea about the testing phase, or I just think the test is running something, that was it. But I was wrong, the regression test actually gave me many ideas about testing the project. Using shell script to do the automatically tests, to compare the running result and the excepted results and gives the "OK" or "FAILED", it is efficient to do the test in this way. In this way, we can easily check our project every time we add new features in it. In total, I learned a lot during working this project and it gives me more preparation to do a harder project.

7.2 Advice for Future Teams

Always start early even if you can't think of an idea or seem to get anywhere with the project. Just attempt to map everything out. As long as you start early when there is very little coursework, then you will be okay. As you have more people in your group, there will be a higher chance of conflicting schedule and deadlines, which means getting this out of the way (similar to FIFO) means you can finish your deadlines for other classes or even socialize. Socializing is actually important because you want to get to know form a group that functions well together and by doing so you should be able to know what each individual's strength and weaknesses are. So get to know your group early!

It also helps to have more people in the group so that someone can take over for another person in case they are busy or not sure about how to implement something. It is near impossible to just have one person designated to a particular job. Everyone has to contribute to documentation, programming, and testing. Giving people a designated job just means they do more of one thing than something else. Due to the limited time constraints, everyone basically needs to contribute to the project in almost every possible way.

The TA and professor is there to help you. Rather than being stuck for days on a particular problem, they might be able to share some insight or guide your project towards the correct direction. Don't be afraid to use them. Contacting them can only make your project better.

Appendix

Gridworld Source Code

Listing 28: scanner.mll

```
{ open Parser } (* Get the token types *)
rule token = parse
(* Whitespace *)
[’ ’ ’\t’ ’\r’ ’\n’] { token lexbuf }

(* Comments *)
| "/*"  { comment lexbuf }

(* Basic tokens *)
(* Parenthesis *)
| ’(’ { LPAREN } | ’)’ { RPAREN }
(* Braces *)
| ’{’ { LBRACE } | ’}’ { RBRACE }
(* Brackets *)
| ’[’ { LBRACKET } | ’]’ { RBRACKET }

| ’;’ { SEMI }   | ’:’ { COLON }
| ’,’ { COMMA }  | ’=’ { ASSIGN }

(* Arithmetic operators *)
| ’+’ { PLUS }   | ’-’ { MINUS }
| ’*’ { TIMES }  | ’/’ { DIVIDE }
| ’%’ { MOD }

(* Logic operators *)
| "==" { EQ }    | "!=" { NEQ }
| ’<’ { LT }     | "<=" { LEQ }
| ’>’ { GT }     | ">=" { GEQ }
| ’!’ {NOT}
| ’&’ {AND}      | ’|’ {OR}
```

Listing 29: scanner.mll

```
(* Keywords *)
| "if" { IF }              | "else" { ELSE }
| "for" { FOR }            | "elif" { ELIF }
| "function" {FUNCTION}    | "return" { RETURN }
| "break" { BREAK }        | "continue" { CONTINUE }
| "while" { WHILE }        | "node" {NODE}
| "main" {MAIN}

(* Type *)
| "int" { INT }
| "bool" { BOOL }
| "char" { CHAR }
| "string" { STRING }

(* Built-in Func *)
| "print" {PRINT}
| "list" {LIST}
| "goto" {GOTO}
| "choose" {CHOOSE}
| "readInt" {READINT}
| "readStr" {READSTR}
| "roll" {ROLL}
| eof { EOF } (* End of file *)

(* Integers *)
| ['0' - '9']+ as lxm { INT_LIT(int_of_string lxm) }

(* Bool *)
| ("true"|"false") as boolean {BOOL_LIT(bool_of_string boolean)}

(* String *)
| '"'(['\000' - '\033' '\035' - '\127']* as str)'"' {STR_LIT(str)}
```

Listing 30: scanner.mll

```
(* ID *)
| ['a' - 'z' 'A' - 'Z']['a' - 'z' 'A' - 'Z' '0' -
    '9' '_']* as lxm { ID(lxm) }
| _ as char { raise (Failure("illegal character "
    ^ Char.escaped char)) }

and comment = parse
    "*/" { token lexbuf } (* End of comment*)
| _ { comment lexbuf } (* Eat everything else *)
```

Listing 31: ast.ml

```
type op =
      Add | Sub | Mult | Div | Equal | Neq | Less
          | Leq | Greater | Geq | Mod | And | Or |
          Not

type scope = Local | Global

type expr =
      Int_Lit of int
    | Bool_Lit of  bool
    | String_Lit of string
    | Id of string
    | Uniop of op * expr
    | Binop of expr * op * expr
    | Assign of string * expr
    | Call of string * expr list
    | Noexpr
```

Listing 32: ast.ml

```
type stmt =
    Print of expr
    | List of expr list
    | Choose of expr list
    | Goto of expr
    | If of expr * stmt list * stmt list
    | While of expr * stmt list
    | Expr of expr
    | Return of expr
    | ReadInt of expr
    | ReadStr of expr
    | Roll of expr

type mytypes =
    Int
    | Bool
    | String
    | Void


type vdecl = {
    vtype : mytypes;
    vname : string;
    vexpr : expr;
}

type param_decl =
    Param of mytypes * string

type fdecl = {
    ftype: mytypes;
    fname : string;
    params : param_decl list;
    body : stmt list;
    }
```

Listing 33: ast.ml

```
type ndecl = {
    nname: string;
    body: stmt list;
}
type program = vdecl list * fdecl list * ndecl
   list
```

Listing 34: parser.mly

```
%{ open Ast %}
%token LPAREN RPAREN LBRACE RBRACE LBRACKET
   RBRACKET SEMI COLON GET COMMA ASSIGN AT
%token PLUS MINUS TIMES DIVIDE PERCENT EXP MOD
%token EQ NEQ LT LEQ GT GEQ NOT AND OR
%token BREAK CONTINUE ELIF ELSE FOR FUNCTION
   RETURN WHILE IF
%token INT VOID BOOL CHAR STRING
%token PRINT GOTO LIST CHOOSE MAIN NODE READINT
   READSTR ROLL
%token EOF

%token <int> INT_LIT
%token <bool> BOOL_LIT
%token <string> STR_LIT
%token <string> ID

%nonassoc NOELSE
%nonassoc ELSE
%nonassoc RETURN
```

Listing 35: parser.mly

```
%right ASSIGN
%left AND OR
%right NOT
%left EQ NEQ LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left MOD
%nonassoc LPAREN RPAREN

%start program
%type <Ast.program> program
%%

program:
    /* nothing */ { [], [], [] }
| program vdecl { let (var, func, node) = $1 in
   $2::var, func, node }
| program fdecl { let (var, func, node) = $1 in
   var, $2::func, node }
| program ndecl { let (var, func, node) = $1 in
   var, func, $2::node }

fdecl:
    mytypes FUNCTION ID LPAREN params_opt RPAREN
       LBRACE stmt_list RBRACE
    {{
    ftype = $1;
        fname = $3;
        params = $5;
        body = List.rev $8
        }}
```

Listing 36: parser.mly

```
ndecl:
  NODE ID LBRACE stmt_list RBRACE
  {{
    nname = $2;
    body = List.rev $4
    }}
  | MAIN LBRACE stmt_list RBRACE{{
    nname = "main";
    body = List.rev $3
    }}

vdecl:
  mytypes ID ASSIGN expr SEMI {{ vtype = $1;
          vname = $2;
          vexpr = $4 }}
mytypes:
  INT {Int}
  | BOOL {Bool}
  | STRING {String}
  | VOID {Void}


params_opt:
      /* nothing */ { [] }
    | params_list   { List.rev $1 }

params_list:
      mytypes ID                 { [Param($1, $2)
         ]}
    | params_list COMMA mytypes ID { Param($3,$4)
        ::$1 }

stmt_list:
      /* nothing */  { [] }
    | stmt_list stmt { $2 :: $1 }
```

Listing 37: parser.mly

```
stmt:
  expr SEMI {Expr($1)}
    | PRINT LPAREN expr RPAREN SEMI { Print($3) }
    | LIST LPAREN actuals_opt RPAREN SEMI{ List($3) }
    | CHOOSE LPAREN actuals_opt RPAREN SEMI{ Choose($3) }
    | GOTO LPAREN expr RPAREN SEMI { Goto($3) }
    | IF LPAREN expr RPAREN LBRACE stmt_list RBRACE { If($3, $6, [])}
    | IF LPAREN expr RPAREN LBRACE stmt_list RBRACE ELSE LBRACE stmt_list RBRACE { If($3, $6, $10)}
    | WHILE LPAREN expr RPAREN LBRACE stmt_list RBRACE { While($3, $6) }
    | RETURN expr SEMI { Return($2) }
    | READINT LPAREN expr RPAREN SEMI { ReadInt ($3)}
    | READSTR LPAREN expr RPAREN SEMI { ReadStr ($3)}
    | ROLL LPAREN expr RPAREN SEMI { Roll ($3) }

expr:
      INT_LIT        { Int_Lit($1) }
    | BOOL_LIT           { Bool_Lit($1) }
    | STR_LIT         { String_Lit($1) }
    | ID                 { Id($1) }
    | NOT expr     { Uniop(Not, $2) }
    | expr PLUS   expr { Binop($1, Add,   $3) }
    | expr MINUS  expr { Binop($1, Sub,   $3) }
    | expr TIMES  expr { Binop($1, Mult,  $3) }
    | expr DIVIDE expr { Binop($1, Div,   $3) }
    | expr MOD    expr { Binop($1, Mod,   $3) }
    | expr EQ     expr { Binop($1, Equal, $3) }
    | expr NEQ    expr { Binop($1, Neq,   $3) }
    | expr LT     expr { Binop($1, Less,  $3) }
    | expr LEQ    expr { Binop($1, Leq,   $3) }
```

Listing 38: parser.mly

```
    | expr GT     expr { Binop($1, Greater,  $3) }
    | expr GEQ    expr { Binop($1, Geq,   $3) }
    | expr AND    expr { Binop($1, And,   $3) }
    | expr OR    expr { Binop($1, Or,   $3) }
    | ID ASSIGN expr   { Assign($1, $3) }
    | ID LPAREN actuals_opt RPAREN { Call($1, $3) }
    | LPAREN expr RPAREN { $2 }

actuals_opt:
    /* nothing */ { [] }
    | actuals_list { List.rev $1 }

actuals_list:
    expr { [$1] }
    | actuals_list COMMA expr { $3 :: $1 }
```

Listing 39: sast.ml

```
open Ast
type t =
    SInt
    | SString
    | SBool
    | SVoid

type sexpr =
      SInt_Lit of int * t
    | SBool_Lit of  bool * t
    | SString_Lit of string * t
    | SId of string * t
```

Listing 40: sast.ml

```
    | SUniop of op * sexpr * t
    | SBinop of sexpr * Ast.op * sexpr * t
    | SAssign of string * sexpr * t
    | SCall of string * sexpr list * t
    | SNoexpr of t

type sstmt =
    SPrint of sexpr
    | SList of sexpr list
    | SChoose of sexpr list
    | SGoto of sexpr
    | SIf of sexpr * sstmt list * sstmt list
    | SWhile of sexpr * sstmt list
    | SExpr of sexpr
    | SReturn of sexpr
    | SReadInt of sexpr
    | SReadStr of sexpr
    | SRoll of sexpr


type svdecl = {
    svtype : t;
    svname : string;
    svexpr : sexpr;
}

type sfdecl = {
    ftype : t;
    fname : string;
    sparams : svdecl list;
    sbody : sstmt list;
}

type sndecl = {
    nname : string;
    sbody : sstmt list;
}
```

Listing 41: analyzer.ml

```
open Ast
open Sast
type symbol_table = {
    mutable parent : symbol_table option;
    mutable variables: (string * svdecl * t) list
       ;
    mutable functions: sfdecl list;
    mutable nodes: sndecl list;
    mutable return_found: bool;
}

type environment = {
    mutable scope : symbol_table;
}

let type_expr (se : Sast.sexpr) : Sast.t =
    match se with
        SInt_Lit(_, t) -> t
      | SBool_Lit(_,t) -> t
      | SString_Lit(_,t) -> t
      | SId(_,t) -> t
      | SUniop(_,_,t) -> t
      | SBinop(_,_,_,t) -> t
      | SAssign(_,_,t) -> t
      | SCall(_,_,t) -> t
      | SNoexpr(t) -> t

let rec check_id (scope : symbol_table) id =
    try
        let (_, decl, t) = List.find(fun (n, _, _
           ) -> n = id ) scope.variables in t
    with Not_found ->
        try let _ = List.find(fun c -> c.nname =
           id) scope.nodes in SString
        with Not_found ->
            match scope.parent with
                Some(parent) -> check_id parent
                   id
              | _ ->  raise Not_found
```

Listing 42: analyzer.ml

```
let rec find_func (scope : symbol_table) f =
    let l = scope.functions in
    try
        List.find(fun c -> c.fname = f) l
    with Not_found -> match scope.parent with
        Some(parent) -> find_func parent f
        | _ -> raise Not_found

let rec find_node (scope : symbol_table) n =
    let l = scope.nodes in
    try
        List.find(fun c -> c.nname = n) l
    with Not_found -> match scope.parent with
        Some(parent) -> find_node parent n
        | _ -> raise Not_found

let rec check_expr_nodes (scope : symbol_table) (e: Ast.expr) =
    match e with
    Noexpr -> SNoexpr(SVoid)
    | Int_Lit(a) -> SInt_Lit(a,SInt)
    | Bool_Lit(a) -> SBool_Lit(a,SBool)
    | String_Lit(a) -> SString_Lit(a,SString)
    | Id(str) ->    SId(str, SString)
    | _ -> raise (Failure("wrong arguments"))

let rec check_expr (scope : symbol_table) (e: Ast.expr) =
    match e with
    Noexpr -> SNoexpr(SVoid)
    | Int_Lit(a) -> SInt_Lit(a,SInt)
    | Bool_Lit(a) -> SBool_Lit(a,SBool)
    | String_Lit(a) -> SString_Lit(a,SString)
    | Id(str) ->    (try
                        let t = check_id scope str in SId(str, t)
                    with Not_found -> raise (Failure ("Unrecognized Id " ^ str)))
```

Listing 43: analyzer.ml

```
    | Uniop(_,_) as u -> check_uniop scope u
    | Binop(_,_,_) as b -> check_binop scope b
    | Assign(_,_) as a -> check_assign scope a
    | Call(_,_) as c -> check_call scope c

and check_uniop (scope : symbol_table) uniop = match uniop with
     Ast.Uniop(op, expr) -> (
         match op with
             Not ->
                 let e = check_expr scope expr in
                 let t = type_expr e in
                 if (t <> SBool) then raise (Failure "Incorrect type for ! ") else SUniop(op, e, SBool)
             | _ -> raise (Failure "Not a uniop")
             )
    | _ -> raise (Failure "Not a uniop")

and check_binop (scope : symbol_table) binop = match binop with
     Ast.Binop(a1, op, a2) ->
         let e1 = check_expr scope a1 and e2 = check_expr scope a2 in
         let t1 = type_expr e1 and t2 = type_expr e2 in
         let t = match op with
             Add ->
                 if (t1 <> SInt || t2 <> SInt) then
                     if (t1 <> SString || t2 <> SString) then raise (Failure "Incorrect types for +")
                     else SString
                 else SInt
             | Sub -> if (t1 <> SInt || t2 <> SInt) then raise (Failure "Incorrect types for  ") else SInt
```

Listing 44: analyzer.ml

```
            | Mult -> if (t1 <> SInt || t2 <>
              SInt) then raise (Failure "
              Incorrect types for * ") else SInt
            | Div  -> if (t1 <> SInt || t2 <> SInt
              ) then raise (Failure "Incorrect
              types for / ") else SInt
            | Mod  -> if (t1 <> SInt || t2 <> SInt
              ) then raise (Failure "Incorrect
              types for % ") else SInt
            | Equal -> if (t1 <> t2) then raise (
              Failure "Incorrect types for = ")
              else SBool
            | Neq  -> if (t1 <> t2) then raise (
              Failure "Incorrect types for != ")
              else SBool
            | Less -> if (t1 <> SInt || t2 <>
              SInt) then raise (Failure "
              Incorrect types for < ") else SBool
            | Leq  -> if (t1 <> SInt || t2 <> SInt
              ) then raise (Failure "Incorrect
              types for <= ") else SBool
            | Greater -> if (t1 <> SInt || t2 <>
              SInt) then raise (Failure "
              Incorrect types for > ") else SBool
            | Geq  -> if (t1 <> SInt || t2 <> SInt
              ) then raise (Failure "Incorrect
              types for >= ") else SBool
            | Or  -> if (t1 <> SBool || t2 <>
              SBool) then raise (Failure "
              Incorrect types for | ") else SBool
            | And  -> if (t1 <> SBool || t2 <>
              SBool) then raise (Failure "
              Incorrect types for & ") else SBool
            | Not -> raise (Failure "! is a unary
               operator.")
        in SBinop(e1, op, e2, t)
    | _ -> raise (Failure "Not an op")
```

Listing 45: analyzer.ml

```
and check_assign (scope : symbol_table) a = match a with
    Ast.Assign(id, expr) ->(
        try(
            let t = check_id scope id in
            let e = check_expr scope expr in
            let t2 = type_expr e in
            if t <> t2 then raise (Failure "Incorrect type assignment.")
            else SAssign(id, e, t))
        with Not_found -> let e = check_expr scope expr in
                                 let t = type_expr e in
                                 let v={ svtype = t; svname = id; svexpr = e}
        in scope.variables <- (v.svname,v,t) :: scope.variables; SAssign(id, e, t))

    | _ -> raise (Failure "Not an assignment")

and check_call (scope : symbol_table) c = match c with
    Ast.Call(id, el) ->
        (try
            let f = find_func scope id in
            let exprs = List.fold_left2 (fun a b c ->
                    let t = b.svtype in
                    let expr = check_expr scope c in
                    let t2 = type_expr expr in
                        if t <> t2
                        then raise (Failure "wrong type")
                        else expr :: a) [] f.sparams el in
                        SCall(id, exprs, f.ftype)
```

Listing 46: analyzer.ml

```
          with
              Not_found ->
                  raise (Failure ("Function not found with name " ^ id)))
      | _ -> raise (Failure ("Not a call"))

let rec check_stmt (scope : symbol_table) (stmt : Ast.stmt) = match stmt with
      Expr(e) -> SExpr(check_expr scope e)
      | Return(e) -> SReturn(check_expr scope e)
      | If(expr, stmt1, stmt2) ->
          let new_expr = check_expr scope expr in
          let t = type_expr new_expr in
          if t <> SBool then raise (Failure "If statement must have a boolean expression")
          else
              let new_stmt1 = check_stmt_list scope stmt1 in
              let new_stmt2 = check_stmt_list scope stmt2 in
              SIf(new_expr, new_stmt1, new_stmt2)
      | While(expr, stmt) ->
          let expr = check_expr scope expr in
          let t = type_expr expr in
          if t <> SBool then raise (Failure "If statement must have a boolean expression")
          else
              let new_stmt = check_stmt_list scope stmt in
              SWhile(expr, new_stmt)
```

Listing 47: analyzer.ml

```
    | Print(e) ->
      let expr = check_expr scope e in
          let t = type_expr expr in
              if (t = SString || t = SInt) then
                  SPrint(expr)
              else raise (Failure "Print takes
                 only type string or int")
    | List(e) ->
      let exprs = List.fold_left (fun a b ->
          let expr = check_expr scope b in
          let t = type_expr expr in
              if t <> SString then
                  raise (Failure "List takes
                     only type string")
              else expr :: a) [] e in
              SList(exprs)
    | Choose(e) ->
      let exprs = List.fold_left (fun a b ->
          let expr = check_expr_nodes scope b
             in
              let t = type_expr expr in
                          expr :: a) [] e in
                  SChoose(exprs)
    | Goto(e) ->
      let expr = check_expr_nodes scope e in
                      SGoto(expr)
    | ReadInt(e) -> SReadInt(check_expr scope e)
    | ReadStr(e) -> SReadStr(check_expr scope e)
    | Roll(e) ->
      let expr = check_expr scope e in
          let t = type_expr expr in
              if (t = SInt) then
                  SRoll(expr)
              else raise (Failure "Roll takes
                 only type int")
```

Listing 48: analyzer.ml

```
and  check_stmt_list (scope : symbol_table) (stml : Ast.stmt list) =
    List.fold_left (fun a s -> let stmt = check_stmt scope s in stmt::a) [] stml

let rec check_stmt_snd (scope : symbol_table) (stmt : Ast.stmt) = match stmt with
    | If(expr, stmt1, stmt2) ->
        check_stmt_list_snd scope stmt1;
        check_stmt_list_snd scope stmt2
    | While(expr, stmt) ->
        check_stmt_list_snd scope stmt
    | Choose(e) -> (
        List.fold_left (fun a b ->
            let expr = check_expr scope b in
                let t = type_expr expr in
                    if t <> SString then
                        raise (Failure ("Choose takes only type string"))
                    else
                        (try
                            let id = match b with
                             String_Lit(a) -> a
                            | Id(str) -> str
                            | _ -> raise (Failure "Wrong expression type in Choose") in
                            let _ = find_node scope id in
                            expr :: a
                        with
                            Not_found ->
                            raise (Failure ("Node not found")))) [] e )
```

Listing 49: analyzer.ml

```
    | Goto(e) -> (
        let expr = check_expr scope e in
            let t = type_expr expr in
                if t <> SString then
                    raise (Failure ("Goto takes only type string"))
                else
                    (try
                        let id = match e with
                         String_Lit(a) -> a
                        | Id(str) -> str
                        | _ -> raise (Failure "Wrong expression type in Goto") in
                        let _ = find_node scope id in
                        [expr]

                    with
                        Not_found ->
                        raise (Failure ("Node not found"))))
    | _ -> [SNoexpr(SVoid)]

and  check_stmt_list_snd (scope : symbol_table) (stml : Ast.stmt list) =
     let _ = List.fold_left (fun a s -> let stmt = check_stmt_snd scope s in stmt::a) [] stml in [SNoexpr(SVoid)]

let rec check_var_type (scope : symbol_table) (v : Ast.mytypes) = match v with
     Ast.Void -> SVoid
    | Ast.Int -> SInt
    | Ast.String -> SString
    | Ast.Bool -> SBool
```

Listing 50: analyzer.ml

```
let process_var_decl (scope : symbol_table) (v :
    Ast.vdecl) =
    let t = check_var_type scope v.vtype in
    let expr = check_expr scope v.vexpr in
    let t2 = type_expr expr in
    if t <> t2 then raise (Failure "wrong type
        for variable initialization")
    else (let v={ svtype = t; svname = v.vname;
        svexpr = expr}
        in scope.variables <- (v.svname,v,t) ::
            scope.variables; v)

let rec check_func_stmt (scope : symbol_table) (
    stml : Sast.sstmt list) (ftype : Sast.t) =
    List.iter (fun s -> match s with
        SReturn(e) ->
            let t = type_expr e in
            if t <> ftype then raise (Failure "
                func return type is incorrect")
                else ()
        | SIf(_, s1, s2) ->
            check_func_stmt scope s1 ftype;
                check_func_stmt scope s2 ftype
        | SWhile(_, s) ->
            check_func_stmt scope s ftype
        | _ -> ()) stml

let rec check_node_stmt (scope : symbol_table) (
    stml : Sast.sstmt list) =
    List.iter (fun s -> match s with
        SIf(_, s1, s2) ->
            check_node_stmt scope s1;
                check_node_stmt scope s2
        | SWhile(_, s) ->
            check_node_stmt scope s
        | _ -> ()) stml
```

Listing 51: analyzer.ml

```
let process_func_stmt (scope : symbol_table) (stml : Ast.stmt list) (ftype : Sast.t) =
List.fold_left (fun a s -> let stmt = check_stmt scope s in
    match stmt with
        SReturn(e) ->
            let t = type_expr e in
            if t <> ftype then raise (Failure "incorrect return type") else
            scope.return_found <- true; stmt :: a
        | SIf(_, s1, s2) ->
            check_func_stmt scope s1 ftype; check_func_stmt scope s2
            ftype; stmt :: a
        | SWhile(_, s) ->
            check_func_stmt scope s ftype; stmt :: a
        | _ -> stmt :: a) [] stml

let process_node_stmt (scope : symbol_table) (stml : Ast.stmt list)=
List.fold_left (fun a s -> let stmt = check_stmt scope s in
    match stmt with
        SReturn(e) ->
            raise (Failure "return statement in node")
        | SIf(_, s1, s2) ->
            check_node_stmt scope s1; check_node_stmt scope s2; stmt :: a
        | SWhile(_, s) ->
            check_node_stmt scope s; stmt :: a
        | _ -> stmt :: a) [] stml

let process_stmt_snd (scope : symbol_table) (stml : Ast.stmt list)=
List.fold_left (fun a s -> let stmt = check_stmt_snd scope s in
    stmt :: a) [] stml
```

Listing 52: analyzer.ml

```
let check_func_decl (env : environment) (f : Ast.fdecl) =
    let scope' = { env.scope with parent = Some(env.scope); variables = []; nodes = env.scope.nodes; functions = env.scope.functions } in
    let t = check_var_type env.scope f.ftype in
    let params = List.fold_left (fun a f -> match f with
        Ast.Param(t, n) ->
            let t = check_var_type scope' t in
            let v={ svtype = t; svname = n; svexpr = SNoexpr(SVoid)} in
            scope'.variables <- (n,v,t) :: scope'.variables; v::a) [] f.params in
    let statements = process_func_stmt scope' f.body t in
    if scope'.return_found then
        let f = { ftype = t; fname = f.fname; sparams = params; sbody = statements } in
        env.scope.functions <- f :: env.scope.functions; f
    else (if f.ftype = Void then
            let f = { ftype = t; fname = f.fname; sparams = params; sbody = statements } in
            env.scope.functions <- f :: env.scope.functions; f
        else raise (Failure ("No return for function " ^ f.fname ^ " when return expected.")))


let check_node_decl (env : environment) (n : Ast.ndecl) =
```

Listing 53: analyzer.ml

```
    let scope' = { env.scope with parent = Some(env.scope); variables = []; nodes = env.scope.nodes; functions = env.scope.functions } in
    let statements = process_node_stmt scope' n.body in
    let n = { nname = n.nname; sbody = statements } in
    env.scope.nodes <- n :: env.scope.nodes; n
let process_func_decl (env : environment) (f : Ast.fdecl) =
    try
        let _ = find_func env.scope f.fname in
            raise (Failure ("Function already declared with name " ^ f.fname))
    with Not_found ->
        if (f.fname = "print" || f.fname = "goto" || f.fname = "list" || f.fname = "choose" || f.fname = "main")
        then raise (Failure "A function cannot have same name as built-in function")
        else
            check_func_decl env f

let process_node_decl (env : environment) (n : Ast.ndecl) =
    try
        let _ = find_func env.scope n.nname in
            raise (Failure ("Node with same name as function " ^ n.nname))
    with Not_found ->
        if (n.nname = "print" || n.nname= "goto" || n.nname = "list" || n.nname = "choose")
        then raise (Failure "A node cannot have same name as built-in function")
        else
```

Listing 54: analyzer.ml

```
            try
                let _ = find_node env.scope n.nname in
                    raise (Failure ("Node already declared with name " ^ n.nname))
            with Not_found ->
                check_node_decl env n

let process_nodes (env : environment) (n : Ast.ndecl) =
    process_stmt_snd env.scope n.body

let process_global_decl (env : environment) (g : Ast.vdecl) =
    try
        let _ = check_id env.scope g.vname in
        raise (Failure ("Variable already declared with name " ^ g.vname))
    with Not_found ->
    process_var_decl env.scope g

let check_program (p : Ast.program) =
    let s = { parent = None; variables = []; functions = []; nodes = []; return_found = false} in
    let env = { scope = s } in
    let (vs, fs, ns) = p in
    let globals = List.fold_left (fun a g -> process_global_decl env g :: a) [] (List.rev vs) in
    let funcs = List.fold_left (fun a f -> process_func_decl env f :: a) [] (List.rev fs) in
    let nodes = List.fold_left (fun a n -> process_node_decl env n :: a) [] ns in
    let _ = List.fold_left (fun a n -> process_nodes env n :: a) [] ns in
    globals, funcs, nodes
```

Listing 55: grid.ml

```
open Printf
open Analyzer

let _ =
let lexbuf = Lexing.from_channel stdin in
let program = Parser.program Scanner.token lexbuf in
let sast = Analyzer.check_program program in
Compile.translate sast
```

Listing 56: compile.ml

```
open Ast
open Sast

    let addTab s = s^"\t"
    let range a b =
        let rec aux a b =
            if a > b then [] else a :: aux (a+1) b  in
        if a > b then List.rev (aux b a) else aux a b;;

    let rec print_list = function
        [] -> ()
        | e::l -> print_int e ; print_string " "; print_list l;;
```

Listing 57: compile.ml

```
    let rec print_expr (e : Sast.sexpr) =
        match e with
        SNoexpr(_) -> print_string ""
        | SId(decl,_) ->  print_string decl
        | SInt_Lit(i,_) -> print_string (string_of_int i)
        | SString_Lit(s,_) -> print_string ("\"" ^ s ^ "\"")
        | SBool_Lit(l,_) -> print_string(string_of_bool l)
        | SAssign(v, e,_) -> print_string (v ^ " = ") ;
          print_expr e;
        | SUniop(o,e,_) -> print_string ("!(");
          print_expr e;
          print_string ")";
        | SBinop(e1, o, e2,_) ->
            print_expr (e1);
            print_string (match o with
          Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
            | Equal -> "==" | Neq -> "!="
            | Less -> "<" | Leq -> "<=" | Greater -> ">" | Geq -> ">=" | Mod -> "%"
            | And -> " and " | Or -> " or "|_ ->"");
            print_expr(e2);
        | SCall(f, expr_list,_) ->
          print_string f ;
          print_string "(";
          let rec print_expr_list_comma = function
            [] -> print_string ""
            | e::[] -> print_expr e
            | e::tl -> print_expr e; print_string ", "; print_expr_list_comma tl
```

Listing 58: compile.ml

```
                in print_expr_list_comma (List.rev expr_list);
                print_string ")";;
    let rec print_expr_noquote (e : Sast.sexpr) =
        match e with
        | SString_Lit(s,_) -> print_string ( s );
        | SId(decl,_) ->  print_string decl;
        | _ -> print_string"";;
    let rec print_stmt (s: Sast.sstmt) (tab:string)= match s with
        SExpr(e) -> print_string tab;(print_expr e); print_string"\n";
     | SPrint(e) ->
         print_string tab;print_string ("print (") ;
         print_expr e ;
         print_string (")\n")
     | SWhile(e, s) ->
         print_string tab;print_string("while (");
         print_expr (e) ;
         print_string ("):\n") ;
         print_string tab;
         print_stmt_wTab s (addTab tab);
        print_string "\n"
     | SReturn(e) ->
        print_string tab;print_string("return ");
        print_expr e
     |SList(e) ->
        print_string tab;
        print_string "print(\"\\n";
        List.iter2 (fun a b-> (print_int a; print_string ": "; print_expr_noquote b;print_string"\\n")) (range 1 (List.length(e))) (List.rev e);
        print_string "\")\n"
```

Listing 59: compile.ml

```
    |SChoose(e) ->
      print_string tab;
      print_string"choice = int(input(\"Enter a choice: \"))\n";
      print_string tab;print_string "\t";
      print_string "while(choice!=-1):\n";
      List.iter2 (fun a b-> (print_string tab;
         print_string "\t\tif (choice==";
         print_int a;print_string "):\n";
         print_string tab;print_string "\t\t\t";
          print_expr b;print_string"()\n")) (
         range 1 (List.length(e))) (List.rev e);
      print_string tab;print_string "\t\telse:\n";print_string tab;print_string"\t\t\tchoice = int(input(\"Invalid Input! Please Re-enter: \"))\n";
    |SGoto(e) ->
      print_string tab;print_expr e;
         print_string"()\n";
    |SReadInt(e) ->
      print_string tab;
      print_expr e;
      print_string " = int(raw_input());\n"
    |SReadStr(e) ->
      print_string tab;
      print_expr e;
      print_string " = str(raw_input());\n"
    |SRoll(e) ->
      print_string tab;
      print_expr e;
      print_string " = randint(1,6);"
```

Listing 60: compile.ml

```
    | SIf(e1, s1, s2) ->
          match s2 with
          [] ->
              print_string tab;
              print_string("if ");
              print_expr e1 ;
              print_string(":\n");
              print_stmt_wTab s1 (addTab tab);
              print_string("")
          |_ ->
              print_string tab;
              print_string("if ");
              print_expr e1;
              print_string(":\n");
              print_stmt_wTab s1 (addTab tab);
              print_string ("\n");
              print_string tab;
              print_string("\telse:\n");
              print_stmt_wTab s2 (addTab tab);
              print_string ""
  and print_stmt_wTab (s:Sast.sstmt list) (tab:
     string) = match s with
     [] -> print_string "";
     | hd::[] ->  print_string tab;print_stmt
        hd tab;
     | hd::tl -> print_string tab ;print_stmt
        hd tab;print_stmt_wTab tl tab;;
  let rec print_type (t: Sast.t)= function
  SVoid -> print_string "void ";
  | SInt -> print_string "int ";
  | SString -> print_string "String " ;
  | SBool -> print_string "boolean ";;
```

Listing 61: compile.ml

```
    let rec print_param (v: Sast.svdecl)= match v with
        |_ -> print_type v.svtype;
            print_string " ";
            print_string v.svname;;

    let rec print_param_list (p : Sast.svdecl list) =
    match p with
        [] -> print_string "";
        | hd::[] -> print_param hd;
        | hd::tl -> print_param hd; print_string ", "; print_param_list tl;;

    let rec print_svdecl  (f : Sast.svdecl) = match f with
        |_ ->
        print_string f.svname;
        print_string "=";
        print_expr  f.svexpr;
        print_string "\n";;

    let rec print_stmt_list (p : Sast.sstmt list) =
    match p with
        [] -> print_string "";
        | hd::[] -> print_string "\t";print_stmt hd ""; print_string "\n";
        | hd::tl -> print_string "\t";print_stmt hd ""; print_string "\n"; print_stmt_list tl;;
```

Listing 62: compile.ml

```
let rec print_sndecl  (f : Sast.sndecl list)(
    v: Sast.svdecl list) = match f with
    [] -> print_string "";
    | hd::[] ->
        print_string "def ";
        print_string hd.nname;
        print_string "(";
        print_string "):";
        print_globals v;
        print_stmt_list (List.rev hd.sbody);
        print_string "\texit()\n";
    | hd::tl ->
        print_string "def ";
        print_string hd.nname;
        print_string "(";
        print_string "):";
        print_globals v;
        print_stmt_list (List.rev hd.sbody);
        print_string "\texit()\n";
        print_sndecl tl v;
        print_string "";

and print_globals (v:Sast.svdecl list) =
    match v with
    [] -> print_string "";
    | hd::[] -> print_string("\n\tglobal ");
       print_string hd.svname; print_string "
       ;\n";
    | hd::tl -> print_string("\n\tglobal ");
       print_string hd.svname; print_string ";
       "; print_globals tl;;
```

Listing 63: compile.ml

```
let rec print_sfdecl  (f : Sast.sfdecl list)(v: Sast.svdecl list) = match f with
    [] -> print_string "";
    | hd::[] ->
        print_string "def ";
        print_string hd.fname;
        print_string "(";
        print_param_list (List.rev hd.sparams);
        print_string "):";
        print_globals v;
        print_stmt_list (List.rev hd.sbody);
        print_string "\texit()\n";
    | hd::tl ->
        print_string "def ";
        print_string hd.fname;
        print_string "(";
        print_param_list (List.rev hd.sparams);
        print_string "):";
        print_globals v;
        print_stmt_list (List.rev hd.sbody);
        print_string "\texit()\n";
        print_sfdecl tl v;
        print_string "";;
let translate (variables, functions, nodes) =
    print_string "from random import randint\n";
    List.iter print_svdecl (List.rev variables);
    print_sfdecl (List.rev functions) variables;
    print_sndecl (List.rev nodes) variables;
    print_string "if __name__ == '__main__':\n\tmain()";
```

Listing 64: Makefile

```
compiler: grid.ml objects
    ocamlc -c grid.ml
    ocamlc -o gw ast.cmo parser.cmo scanner.cmo
       compile.cmo analyzer.cmo grid.cmo

objects: scanner parser generator
    ocamlc -c ast.ml sast.ml parser.mli scanner.
       ml parser.ml analyzer.ml compile.ml
generator: analyzer.ml compile.ml
parser: parser.mly
    ocamlyacc -v parser.mly
scanner: scanner.mll
    ocamllex scanner.mll
.PHONY: test
test: compiler
    ./testall.sh

.PHONY: clean
clean:
    rm -f *.py parser.mli scanner.ml parser.ml
       parser.output *.cmo *cmi test-*.py  test-*.
       i.* grid gw *~
```

Gridworld Project Log

```
commit 413b1cfb551061b2bd3ea6bc975b2c396ec70edb
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Tue Dec 22 20:08:23 2015 -0500

    zZz added new README, Makefile, organized everyth

commit 0ed84e43331b064dd5e39e5e3937df27d27975e7
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Tue Dec 22 19:27:16 2015 -0500

    added the tests. Getting ready for submission.

commit fc905e9613e9882daa6bfe8e3712d3831c603f06
```

```
Author: Loren <lorenweng@gmail.com>
Date:   Mon Dec 21 01:08:10 2015 -0800

    fixed bugs in pokeySim

commit e9310b608ad214deaac5fad6746f99039a728ec2
Author: Loren <lorenweng@gmail.com>
Date:   Mon Dec 21 00:37:24 2015 -0800

    pokeysim now working

commit bf94990fd811f3a9000cdbe02e724ca559e692c6
Merge: fb770b8 f500e48
Author: Loren <lorenweng@gmail.com>
Date:   Sun Dec 20 22:37:42 2015 -0800

    Merge branch 'master' of https://github.com/andyph666/gridworld-proj

commit fb770b89a7f9a7d661379ba5a7d9b04268f7ac78
Author: Loren <lorenweng@gmail.com>
Date:   Sun Dec 20 22:29:00 2015 -0800

    fixed tabbing issue

commit f500e48b9c0d2430ed8c844ad0e97ceec7f170ba
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Mon Dec 21 00:18:58 2015 -0500

    edit mkfile to rem more tmp file. Rm arraytests

commit 643ec1fcd17c924a20022b2641028d5ec4ec1d9b
Author: Loren <lorenweng@gmail.com>
Date:   Sun Dec 20 20:14:56 2015 -0800

    compile.ml changes

commit afd3d041c3fa57dbd08ee66d181535f45f136a32
Author: Loren <lorenweng@gmail.com>
Date:   Sun Dec 20 20:08:33 2015 -0800

    added textAdventure.gw

commit cc6613a417597a08aa1e951711b3e952ee87aa3b
Author: Loren <lorenweng@gmail.com>
Date:   Sun Dec 20 19:54:53 2015 -0800

    test3.gw file

commit 4b8dc0d21c2abf7351a8cd4c1bd53fb3d7091537
Author: Loren <lorenweng@gmail.com>
Date:   Sun Dec 20 19:51:10 2015 -0800

    test3 working

commit b39902af6f5091c92ec51be7c8a46405c4980909
Merge: ca8a109 4caba07
Author: Loren <lorenweng@gmail.com>
```

```
Date:   Sun Dec 20 14:59:00 2015 -0800

    Merge branch 'master' of https://github.com/andyph666/gridworld-proj

commit ca8a109188c88c2c485241dc7a2acb79e255f4b0
Author: Loren <lorenweng@gmail.com>
Date:   Sun Dec 20 14:58:42 2015 -0800

    got test2 working

commit 4caba0714a55ff7663197ac25ab682bb27f33737
Author: weng-kevin <wengkevin2002@gmail.com>
Date:   Sun Dec 20 14:58:01 2015 -0800

    made changes to analyzer

commit fc0e01ad2c94b4a2038167fd53e3a53ddbfad444
Author: weng-kevin <wengkevin2002@gmail.com>
Date:   Sat Dec 19 20:39:45 2015 -0800

    update analyzer

commit 70d1b8706487dfd365f02938c82c0824c45c6fda
Author: weng-kevin <wengkevin2002@gmail.com>
Date:   Sat Dec 19 20:18:20 2015 -0800

    edit analyzer

commit 040efddd0c960518af64ccb13da93822bd1748dc
Author: Loren <lorenweng@gmail.com>
Date:   Sat Dec 19 20:17:21 2015 -0800

    more compiler fixes

commit 58f943fac07066bec97eb930f9b527e37e2e563f
Author: Loren <lorenweng@gmail.com>
Date:   Sat Dec 19 16:54:09 2015 -0800

    changes to scanner and parser for missing tokens

commit 80efdebf85eed0cfa259ee10b3ba9cc3544656e3
Merge: 2ac019f a512769
Author: Loren <lorenweng@gmail.com>
Date:   Sat Dec 19 16:50:56 2015 -0800

    merging testall
    Merge branch 'master' of https://github.com/andyph666/gridworld-proj

commit 2ac019ff0e867cea3c1b6fb0c029348f1e8fb4ee
Author: Loren <lorenweng@gmail.com>
Date:   Sat Dec 19 16:50:30 2015 -0800

    changes to scanner and parser for missing tokens

commit a512769b0ff16507b7cfa08f9a3a73fc2878ccc8
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sat Dec 19 19:40:13 2015 -0500
```

```

    removed unicode chinese from testall for latex pdf

commit c79bfb697087a81a28bf992e81416d8c594e5c3d
Author: Loren <lorenweng@gmail.com>
Date:   Sat Dec 19 16:14:17 2015 -0800

    changed more things

commit ca275a31e1161e00f80a7e9df824dde95f33b01f
Author: Loren <lorenweng@gmail.com>
Date:   Sat Dec 19 14:49:28 2015 -0800

    kevin's analyzer changes

commit 576756bc2e74058aed5480bb96e2c08033e3aab0
Author: Loren <lorenweng@gmail.com>
Date:   Sat Dec 19 14:21:40 2015 -0800

    hello world is working now kinda

commit f9c2d881f15cf33293db0cd9ef173fe1cb3974a3
Author: Loren <lorenweng@gmail.com>
Date:   Fri Dec 18 09:57:45 2015 -0800

    grid to sast

commit f4ab255201c5c099a28aba12434cea7e8076d6e8
Author: Loren <lorenweng@gmail.com>
Date:   Fri Dec 18 09:44:01 2015 -0800

    fixed soem bugs in analyzer sast and compile.ml

commit 97be60f5a2c227ba51a279b509506766178ae74f
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Fri Dec 18 11:54:57 2015 -0500

    CODE NOT WORKING. Shift/reduce conflicts gone

commit 050f952dc8d29666caa64ac57db1217a8890fbcc
Merge: 664aeeb 812e82d
Author: weng-kevin <wengkevin2002@gmail.com>
Date:   Thu Dec 17 19:45:05 2015 -0800

    Merge branch 'master' of https://github.com/andyph666/gridworld-proj

commit 664aeebb33844af422c57002f84d69d308ec2118
Author: weng-kevin <wengkevin2002@gmail.com>
Date:   Thu Dec 17 19:44:04 2015 -0800

    Added built-in functions

commit 812e82d09346c0c1f636856267653c00c2960318
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Thu Dec 17 21:27:02 2015 -0500

    added more test functions
```

```

commit d27d87c6946059900916413fd81a7181ef2f1f49
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Thu Dec 17 18:12:13 2015 -0500

    updated makefile

commit 7f58f0510f0fcad277653a4fcb1201eb381f7eaf
Merge: 0c69d71 0d3fe6b
Author: weng-kevin <wengkevin2002@gmail.com>
Date:   Thu Dec 17 15:05:03 2015 -0800

    Merge branch 'master' of https://github.com/andyph666/gridworld-proj

    Conflicts:
        gridworld-src/ast.ml
        gridworld-src/grid.ml
        gridworld-src/parser.mly

commit 0c69d71ec4bb52148cf79c8f5789019a2c8146b2
Author: weng-kevin <wengkevin2002@gmail.com>
Date:   Thu Dec 17 14:52:34 2015 -0800

    Added semantic analyzer and sast, edited parser ast scanner

commit 0d3fe6b7aa0591106f560e6601f5480d60adab97
Author: Zikai Lin <jotaku@dyn-129-236-216-222.dyn.columbia.edu>
Date:   Wed Dec 16 20:19:05 2015 -0500

    added tests

commit 5aa5551bf3a7b74194dc6bc547d42f035ffdc40a
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sat Dec 12 12:17:34 2015 -0500

    added microc incase we need to reference it

commit 6499de41e0c6b0db988d1819d1c854bf5c441edb
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Fri Dec 11 21:27:08 2015 -0500

    Makefile update

commit d7763123dd2c76a23d8ac4805944021f305771e2
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Fri Dec 11 21:22:09 2015 -0500

    fixed linear regression tester and Makefile. Renamed some files.

commit 221dd28b7de7749192f4cc761846a0f520a0a7a1
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Tue Nov 17 13:55:55 2015 -0500

    removed unnecessary files

commit 248e14926f93a145d63fa1a4b8094852d7887d5a
Author: Andrew Phan <ap3243@columbia.edu>
```

```
Date:   Tue Nov 17 13:54:05 2015 -0500

    linear regression tester NOT WORKING

commit 9ae9ca2fd1dc36e46b484871e64a14d825d03ea6
Author: Loren <lorenweng@gmail.com>
Date:   Sun Nov 15 18:51:06 2015 -0800

    added enough functionality to get gcd to work

commit b8e513090900ea91342ea7f3236a62f018192679
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sun Nov 15 20:16:08 2015 -0500

    added makefile

commit d7a59ce595c9e1ee3b39b29974ab35f2783fca69
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sun Nov 15 19:42:43 2015 -0500

    deleted stuff

commit 3abd48bd7eb7eb8689d44820c7308a2214346dba
Merge: 48f08b6 0e07ac9
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sun Nov 15 19:37:25 2015 -0500

    Merge branch 'master' of github.com:andyph666/gridworld-proj

commit 48f08b6ccc6123f2d8f5cd0e26a30bb6f8eae38c
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sun Nov 15 19:37:10 2015 -0500

    added playgw

commit 0e07ac94a65f496dda4fee3199069d2fb2721f09
Author: Loren <lorenweng@gmail.com>
Date:   Sun Nov 15 16:30:19 2015 -0800

    added vdecl stuff

commit 3764d85698212dbaf1d3f42e587990868b0183c8
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sun Nov 15 19:21:22 2015 -0500

    removed objects

commit 722a222e0254e7f1aea412c14a58b1082c40da41
Author: Loren <lorenweng@gmail.com>
Date:   Sun Nov 15 15:52:34 2015 -0800

    added working mod (%) functionality

commit 06d613c4b21e01a6b68fca1d8729b555eefb29e3
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 18:26:45 2015 -0500

```

```
    removed references to vdecl

commit 59a7a9342867e035fdbcc005297cc824c39e9255
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 18:24:38 2015 -0500

    parser removed vdecl

commit 7c5c7efdaa35db003e485f5019791e6f0420c263
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 18:21:15 2015 -0500

    fix

commit dcceaea873651cbc4d802c877e6ab7209ef63818
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 18:18:28 2015 -0500

    mod fix

commit f2a6ab9fbf4ecce00b16011db54954eae780d6d1
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 18:17:00 2015 -0500

    test mod

commit 2429f6efcb2be30adb3e05795d0281e195bafbb9
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 18:15:50 2015 -0500

    added mod(%) and vdecl fixes

commit 87bdbfa863a556fa264fe44c559b83e7090f5b1b
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 18:04:36 2015 -0500

    vdecl fixes

commit 04160cfb986d3e7acbfff35ca1d3673d144e2b33
Merge: 614fe41 7eb66f7
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sun Nov 15 18:03:26 2015 -0500

    Merge branch 'master' of github.com:andyph666/gridworld-proj

commit 614fe4190819546f85b34a2e23d566f7e1f1583c
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sun Nov 15 18:03:01 2015 -0500

    filenames

commit 7eb66f73da7b54762b974ef875285d0ef59ac14c
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:54:26 2015 -0500

    added vdecl stuff

```

```
commit ad004cac21369f7f27da10df8343977f57f0a0d7
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:51:43 2015 -0500

    fix2

commit 48ba6dedc841d39861bd7bbf9b3ed7b6b555725e
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:51:00 2015 -0500

    fix

commit f4d0fec7598b26a0c1d3afe059e40b777af291e3
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:49:41 2015 -0500

    added more stmt

commit e1584072d27c8fe2c2c9c063a89879c29898e4dd
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:46:51 2015 -0500

    paras -> params

commit 7c61eacf40b417894f77502c3ce68d210d873104
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:45:04 2015 -0500

    changed from bodies to program

commit 728139d3108c530e77efcff3aafd3ce9ce544033
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:42:45 2015 -0500

    change paras to params

commit e7a35088235c35e2af7294f71cb18bacca9f65fc
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:40:16 2015 -0500

    test ast.ml changes

commit 875e6b4eaf60452455e96bda00c0cf848c904742
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:38:31 2015 -0500

    reverted changes

commit fe3634f1123a9230e7a6fbb6a4a533190ca5a7b3
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:34:28 2015 -0500

    test push

commit c2a75524b34042af8b3a06d3f3559eecdaa33bff
Author: lorenweng <lorenweng@gmail.com>
Date:   Sun Nov 15 17:31:43 2015 -0500
```

```

    ast.ml and parser.ply changes

    added stuff like if/while statement handling

commit f675cb612cbe4b65fc141fc3f5024b97a7bf8ddf
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sat Nov 14 15:24:18 2015 -0500

    gridworld first commit

commit 697ece706e064330717a8d999de6ce9fe6429161
Author: Andrew Phan <ap3243@columbia.edu>
Date:   Sat Nov 14 15:23:01 2015 -0500

    first commit
```

code/git.c