BoredGames Final Report

A Language for Board Games

Brandon Kessler (bpk2107) and Kristen Wise (kew2132)

Table of Contents

1. Introduction

BoredGames is a language designed for easy implementations of turn-based board games. It allows for the programmer to quickly and efficiently setup a board, establish rules for the game, and define how each turn should be played. All three of these tasks are spilt into different sections which are then integrated together by the BoredGames compiler. BoredGames compiles into Java code, which then is compiled into executable code.

1.1 Motivation

Traditionally developing board games in languages such as C++ or Java is a very time consuming task. A great deal of setup is required and many dependencies need to be dealt with before the rules of the game can even start to be coded. Additionally, much of the code is not reusable from one game to another. BoredGames aims to fix these problems by providing the tools to quickly develop game boards, pieces, and players without the need to deal with the many intricacies that normally go along with these tasks so that the developer can jump right into defining the rules and procedures of the game. This simplified process also allows the user to easily make changes in the game without altering much of the code. We also will allow the user to easily traverse the board and make changes. Board games are often very complicated to create on a computer and we aim to simplify this process as much as possible while still providing the ability for users to develop a wide range of games through the powerful and diverse set of tools within our language. Some simple games that can be implemented in BoardGames include Tic-Tac-Toe, Checkers, and Chess.

2. Language Tutorial

2.1 Structure

All BoardGames programs need to have the following structure:

```
Setup{
	/* General setup such as board, pieces and players goes here*/
}
Rules{
	/* the rules of the game go here. Examples of this are invalid move
	 checks and  win condition checks.*/
}
Play{
	/* How a player’s turn should proceed from start to end goes here*/
}
```

The Setup section must contain a new Board declaration; meanwhile the Rules and Play sections do not require any content. Everything beside control statements (if, else and loops) must end in a semicolon. The program will first evaluate the Setup section, then loop through the Play section continuously until the EndGame function (see below) is reached.

2.2 Declaring Board, Players, and Pieces

Setting up the Board, Players and Pieces must be done in the Setup sections. The following is how to setup each of these three items.

Setting up a Board requires two parameters, the number of rows and the number of columns as follows:

```
new Board(number of rows, number of columns);
```

Adding a Player to the game just requires specifying the name of the Player as a string literal. Note the order you add Players affects the CurrentPlayer and the NextPlayer functions (see 2.3). It can be done as follows:

```
new Player(name);
```

Adding Pieces to the game can be done in two ways. The first way just requires the owner of the Piece as a string literal, the name of the Piece as a string literal, and the number of this Piece that should be created as an int literal. All Pieces created in this manner are placed in the owner's inventory. This can be done as follows:

```
new Pieces(owner, name, number)
```

The other way to add a Piece to the game is in addition to the three parameters above specifying a point value as an int literal after the previous three parameters this can be done as follows:

```
new Pieces(owner, name, number, point value)
```

2.3 Built In Functions

There are a few built in functions available to the user. These functions exist to make game actions, such as adding or moving a piece as simple as possible.

Output takes a string as an argument and displays it to the user in the terminal

```
Output(string);
```

Input takes a variable as its argument, gets command line input from the user and stores that input in the argument variable.

```
Input(input_location);
```

move takes a Piece, a x location, and a y location and takes that specific Piece on the Board and moves it to the coordinate given. If multiple Pieces fit the Piece description the first one reached will be moved.

```
move(Piece, x, y);
```

add takes the specified Piece, which is currently in the current Player's inventory and places it on the Board at the x and y coordinates given. If multiple Pieces fit the Piece description the first one reached will be added.

```
add(Piece, x, y);
```

EndGame prints the message supplied in the string and stops the repeated calls to Play. It ends the program entirely.

```
EndGame(string);
```

CurrentPlayer returns the Player whose turn it currently is. To move to the next Player in the list use the NextPlayer function (see below).

```
CurrentPlayer();
```

NextPlayer moves the return of CurrentPlayer to the next Player in order of Player declarations in the Setup section. Note NextPlayer does not have parentheses following it.

```
NextPlayer;
```

2.4 A Sample Program in BoredGames

Below is a working program in BoredGames. This program creates a 3x3 Board and displays Hello World.

```
Setup{
     new Board(3,3);
}
Rules{
}
Play{
     EndGame("Hello World");
}
```

2.5 Compiling and Running a BoredGames Program

A Makefile is provided for easy compilation of our BoredGames compiler. Simply type make in the correct directory and the compiler will be generated (named compiler). To run a BoredGames program simply type ./run.sh *input_file* where input_file is the .board file containing your BoredGames code. The program will be run through our compilers giving us java code and then compiled using the javac compiler and run using java. Currently our program uses java 6.

2.6 Rules

So far we have looked at the Setup and Play sections. The Rules section is different in that the only way for a specific rule to be used is to reference it in the Play section by its rule id. Each rule can be as many statements as you want, but it must return a bool. A rule can be referenced anywhere in the Play section. Each rule must end in a semicolon to indicate the end of that specific rule. Additionally, a rule can reference any variable declared in the top level scope of the Play section, this allows for easy use of variables without needing to pass them around (More on this in the Scope section of the LRM). The following is a simple rule declaration:

```
rule r1: return true;;
```

2.7 Variables

To declare a variable in BoredGames use the following format where Type is one of the variable types in our language defined in the LRM:

```
Type Variable_ID;
```

or you may assign a value to variable as you declare it by using the following syntax:

```
Type Variable_ID = value;
```

2.8 Control Flow

There are two types of control flow in BoredGames. The first is if and else statements. The if statement does not require an else to follow, but the else statement must always follow an if. The if statement must contain a bool. The following is the syntax used for if and else statements.

```
if(bool){

}

else {

}
```

There is also the loop loop in our language. loop either takes a bool statement or an int followed by a colon followed by an int as its parameters. This second format loops through 2nd int – 1st int + 1 times. The following is syntax for both versions:

```
loop(2 > myInt){
      Output(“counter”);
      myInt = myInt + 1;
}
loop( 1: 3) {
      Output(“counter”);
}
```

3. Language Reference Manual

3.1 Lexical Conventions

3.1.A Comments

BoredGames allows multiline comments. The syntax is as follows:

Comment Symbol	*Example*
/* */	/**Comment goes here**/

An example of the comment on multiple lines is:

```
/* this is an example
        on two lines*/
```

3.1.B Tokens

BoredGames has six types of tokens: identifiers, keywords, constants, string literals, operators, and separators. Some whitespace is required to separate adjacent identifiers, keywords, and constants.

3.1.C Identifiers

An identifier is a sequence of letters, digits, and underscores. An identifier must start with a letter or an underscore. Letters include the ASCII characters A-Z, and a-z. Digits are the characters 0-9. BoredGames is case sensitive.

3.1.D Keywords

The following identifiers are reserved as keywords, and cannot be redefined:

Keyword	*Description*
bool	Boolean Data Type
float	Double Data Type
int	Integer Data Type
string	String Data Type
Board	The Game Board
Player	The Players in the Game
Pieces, piece	The Game Pieces, an individual piece
Setup	Where Pregame Setup Occurs
Rules	Where the Rules of the Game are Defined
Play	Where how a Turn will Proceed is Specified
Rule	A Specific Rule of the Game
if, else	Conditional expression, else is optional
loop	A Conditional Repetition of a Block of Code
return	A Value to be Returned from a Rule to its Caller
NextPlayer	Moves the Current Player to Whoever is Next
true	Boolean Constant
false	Boolean Constant

3.1.E Constants

There are a few types of constants in BoardGames:

Type	Constant
Integer	Sequence of digits [0, 9] can be negative
Boolean	True, False
Float	Integer part followed by a decimal part each of which are composed of a sequence of digits [0, 9]

3.1.F String Literals

A sequence of ASCII characters surround by double quotes. For example:

```
    “hello world”
```

3.1.G Operators

BoredGames has comparison, arithmetic, boolean, and declaration operators. The syntax of these operators can be found later in the manual.

3.1.H Separators

Separator	*Description*
,	Separates Elements in a List
;	Ends the Current Line

3.2 Meaning of Identifiers

In BoredGames an identifier is a keyword, a rule, or a name for a variable. The syntax for identifiers is defined in section 3.1.C

3.2.A Variables

A variable is an identifier that has a constant value assigned to it. Each variable must have a type defined by the user. Using a variable within the scope of its declaration will get the value bound to the variable.

3.2.B Rules

Rules are essentially functions without parameters that are called and must return either true or false. Each rule has an identifier bound to it so that the rule can be referred to in the Play section of the program.

3.2.C Scope

The scope of a variable identifier varies depending on which section the variable is defined in. In the Setup section all variables are local and their scope begins at the end of their declaration and continues until the end of the current block (a block is defined by a { followed by code and ending with a }. In Play section all variable declarations on the top level block are in the scope of the rest of the Play section and all of the Rules section. Any variable defined within a sub-block inside of Play is local to its block. In the Rules section all new variable declarations are local to their rule and block.

3.2.D Basic Types

Basic Type	*Description*
bool	Boolean Type
Int	Integer Type
float	A C like Double Type
string	C Like String Type

3.2.E Derived Type

There is a derived type in BoredGames. It is the rule type. rule is a java like function that takes no parameters and must return a boolean. A rule can only be defined in the Rules section of the program. The syntax for a rule is seen in section 2.6.

3.3. Expressions

For our purposes an lvalue is an identifier with a suitable type that is on the left hand side of an expression.

3.3.A Primary Expressions

primary-expression:

constant

identifier

string

(expression)

A constant is a bool, float, or int literal, as opposed to a variable. An identifier has the type specified at its declaration. A string is characters surrounded by “ ”. A parenthesized expression has the same type and value as the original expression.

3.3.B Postfix Expressions

Operators are bound to expressions from left to right.

postfix-expression:

primary-expression

postfix-expression [expression]

where postfix-expression is either Player or Pieces

postfix-expression (argument-expression-list$_{optional}$)

where postfix-expression is a function that is being called

postfix-expression . identifier

where postfix-expression is an either of the two expression above and identifier is another library function.

postfix-expression ++

where postfix-expression is an int or float

postfix-expression --

where postfix-expression is an int or float

argument-expression-list:

assignment-expression which is a variable id or a constant

argument-expression-list , assignment-expression

3.3.C Array References

Although arrays may not be directly defined in BoredGames, there are multiple array structures within Board, Pieces, and Player that need to be accessed. An expression in square brackets after the Pieces or Player identifier indicates the access of an array reference. The expression inside the brackets must be of integer type between 1 and the size of the array, signifying the index of the array being accessed. An expression in square brackets after Board identifier indicates the access of an array reference. The expression in the brackets must be in the following format (integer, integer) in order to access the x and y coordinates of the Board. For example:

```
Board[(1,2)]    /*accesses the Board at location x=1,y=2*/
```

3.3.D Function Calls

Although functions may not be defined in BoredGames, there are predefined library functions available. A function call is an identifier followed by parenthesis and takes a variable-length set of arguments separated by commas. There are specific examples in section 2.3. The argument list length depends on the particular input required for a given function, and can be zero. Some

BoredGames functions have are overloaded and accept different input types depending on the length of the provided argument list. The arguments passed to a function should match the function parameters in type and should be within scope of the function's call. If these conditions are not met, it will result in an error. Though it is not necessary for the return of a library function to be used, if it is used the type must match where it is being used.

3.3.E Object Structure Reference

A postfix expression followed by a dot and identifier is used to access details of the items (Board, Pieces, and Player defined in greater detail in later sections) in BoredGames. The specific members are unique to each item as seen below. The type of the object member can be of a structural, derived, or basic type, and can be assigned to an lvalue with a matching type or altered according to the specific type's rules. This is further discussed in the Board, Player, and Pieces sections.

3.3.F Postfix Incrementation and Decrementation

An int or float followed by a ++ or -- operator results in a int or float of the same algebraic type as the original expression. The value of the expression is incremented or decremented, respectively, by 1. The expression must be an lvalue, as opposed to a constant.

3.3.G Logical Operators

Logical operators are performed on Boolean values and return Boolean values.

logical-expression: Boolean logical-operator Boolean

where Boolean is either a constant boolean or a variable id of type bool

Operator	*Explanation*
&&	Logical AND: True if both sides are True; False otherwise
||	Logical OR: True if either side is True; False if both sides are False

3.3.H Numerical Operators

These numerical operators are binary operators. Numerical operators return type float if a float is used in either expression otherwise they return an int. In order of precedence:

Operator	*Description*
*	Multiplication
/	Division
+	Addition

-	Subtraction

Numerical operators group left-to-right, and can be performed on int or float data types. Integer division result in an integer with the remainder discarded.

3.3.I Relational and Equality Operators

Relational and equality operators return bool values. If the expression is true, the return value is true; if the expression is false, the return value is false.

relational/equality-expression:

rel/eq-expression rel/eq-operator rel/eq-expression

In order of precedence:

Operator	Description
<	Less Than
>	Greater Than
<=	Less Than or Equal to
>=	Greater Than or Equal to
==	Equal to
!=	Not Equal to

Relational operators, the first four listed, bind tighter than equality operators, the last two listed. So when used in conjunction:

```
a>b == b>d      /* is equivalent to*/

(a>b) == (b>d)
```

3.4. Declarations

A declaration may or may not have a value assignment. Piece and string types need to be declared separately from their assignments, howeverExamples of this are as follows:

```
int x;          /* either are acceptable */

int x = 1;
```

but

```
piece p = Player.inventory();    /* not allowed */

piece p;                         /* must declare */
```

```
p = Player.inventory();          /* then assign*/
```

3.4.A Type Specifiers

type-specifier:

int

double

bool

string

An identifier has exactly one type. The type specifier precedes the identifier.

3.4.B Loop Declaration

loop (*expression)* { *statement* }

The expression in the loop parenthesis evaluates to a bool, which is evaluated before the body of the loop is executed. The statement in the braces is a sequence of expressions that takes place if the expression bool evaluates to true. The expression will be repeatedly tested after each iteration of the loop until it fails. Then, the loop breaks and the next portion of the program runs.

3.4.C Through Statement

The colon operator acts as an iterator when located between two integer constants or identifiers. It starts at the left-hand value and increments by 1 until it reaches the right-hand value. The colon can be used inside a loop expression.

constant:constant

constant:identifier

identifier:constant

identifier:identifier

For example:

```
loop(1:6) {}          /* constant:constant */
int a=1;
int b=6;              /* identifiers must be int */
```

```
loop(a:b) {}          /* identifier:identifier */
```

3.4.D Selection Statements

if (*expression*){ *statement }*

if (*expression*){ *statement* }else{ *statement* }

The expressions for all if statements must evaluate to a bool value, through relational and equality operators can be used to translate arithmetic types into Boolean, i.e. (5<6) evaluates to true. The if statement checks the expression, executes the statement, and continues to the next line of the program. The if, else statement blocks insist that either one statement or the other is executed before the next section of the program is run.

3.5. Program Components

Program components are the three items that make up BoredGames programs; Board, Player, and Pieces. The three sections; Setup, Rules, and Play also fall under this category. These six things combine to make BoredGames a unique and powerful language.

3.5.A Board Item

3.5.A.1 Board Declaration

The Board is structurally a matrix of any size. The parameters passed are the number of rows and columns that will be needed. Here is the syntax for the creation of the Board:

```
new Board(number-of-rows, number-of-columns);
```

3.5.A.2 Board Access

When Pieces are created, they can be set to a certain player's *inventory* (off of the Board) or a Player's *onBoard* (at a Board coordinate). This is discussed in the Piece Item section. BoredGames has two library functions *add(...)* and *move(...)* (discussed in Library Functions) that modify the location of Pieces on the Board. Board also has a function that can be used in the Rules section of the program to check if a position has any Pieces. It returns a bool, true or false. The syntax is as follows

```
Board[(x_location,y_location)].unoccupied();
```

Every location on the Board has a list of Pieces on that location as well, and it can be accessed by writing `Board[(x_location,y_location)].Pieces[i];` where i is the index location of the Piece. Index locations correspond to the order the Pieces were added to that location of the Board.

3.5.B Player Item

3.5.B.1 Player Declaration

A Player item is created for each of the game users. When Pieces are created, they are assigned to a Player and given titles and, optionally, point values.

```
new Player(player-name);
```

3.5.B.2 Player Access

Within Players there are *inventory* and *onBoard* lists that distinguish between Pieces that belong a particular Player off and on the Board. These lists can be accessed using the inventory and onBoard calls whose syntax is below. These two calls return a Piece. Player *inventory* and *onBoard* sections are accessed similarly to how Pieces is accessed in the Board Item section. Pieces are stored within inventory and onBoard in the order they are added. Methods of accessing Pieces are further described in the Pieces Item section.

To access the CurrentPlayer's inventory or their onBoard lists through indices the following syntax may be used:

```
CurrentPlayer().inventory(index);

CurrentPlayer().onBoard(index);
```

If not argument is provided the first item in the inventory or in onBoard is provided.

To access the CurrentPlayer's inventory or their onBoard lists through the name of a piece the following syntax may be used. Note that the first instance a Piece with a matching name to *piece-name* will be returned.

```
CurrentPlayer().inventory(piece-name);

CurrentPlayer().onBoard(piece-name);
```

The CurrentPlayer's name can be gotten using the name call which will return the Player's name as a string. The syntax is as follows:

```
CurrentPlayer().name();
```

Additionally, anywhere `CurrentPlayer()` or `CurrentPlayer().name()` is used, `Player` (without any brackets or parenthesis) can be used, as it means the current player.

3.5.C Pieces Item

3.5.C.1 Pieces Declaration

Game Pieces are created with the Piece Item which specifies the owner of the Piece, the name of the Piece, and the number of pieces with these same properties. Three required fields are player-name (type string), piece-name (type string), and number-of-pieces (type int). Optionally a Piece declaration can also specify a point-value (type float or int) for the Piece. The following are Pieces Declaration syntax:

```
     new Pieces(player-name, piece-name, number-of-pieces);

     new Pieces(player-name, piece-name, number-of-pieces, point-
value);
```

3.5.C.2 Pieces Access

Pieces are stored and can be accessed from either the Board or Player items. BoredGames handles the Pieces' underlying connections between the Board and Player, so there are no updates required. Library functions, such as move and add, take care of updating the current game state so the programmer can focus on defining rules of the game. A Piece can be accessed through Player *inventory* or *onBoard* as seen in the Player Item section. When a Piece is in the a Player's inventory its location is considered 0,0. This is useful for when we try to access the location of a particular Piece. Pieces can also accessed through the Board as seen in the Board Item section. There is no direct access to the Pieces, anytime a user wishes to access a Piece they must either go through Board or Player. This is done to keep the overall idea of Pieces belonging to a Player from becoming diluted.

3.5.C.3 Pieces Calls

Pieces have information stored in them, and this information can be accessed the following calls. Once a Piece is accessed, by any of the method mentioned in the Board Item and Player Item sections, these calls can be applied:

The first call we will look at is owner. owner returns a string containing the name of the owner of the Piece the syntax is as follows where piece-access is whichever of the Piece access methods the programmer chooses to use:

```
     piece-access.owner();
```

The next call we will look at is name. name returns the name of the piece as a string. The syntax is as follows:

```
piece-access.name();
```

Third we will look at the point call. point returns the point value of the piece as either a float or int depending on what the point value was defined as. If the point value was never defined 0 is returned.

```
piece-access.point();
```

Lastly, we will look at the locationx and locationy calls. These calls return the pieces x location on the Board and its y location on the Board respectively both as ints. The syntax is as follows:

```
piece-access.locationx();

piece-access.locationy();
```

3.5.D Setup Section

The Setup section of a BoredGames program is where the initial Board , Players and necessary Pieces for the game are defined. The section is enclosed in braces. The Setup section should follow the following syntax

```
Setup { /*statements go here */ }
```

3.5.E Rules Section

The Rules section of a BoredGames program is a series of rules statements. A rule is declared using the rule keyword followed by an identifier, starting with a capital “R”, and a colon followed by a list of statements. Each rule must return a bool and should end in a semicolon. Each individual rule also has access to variables initialized in the top level scope of the Play section. rules can be used in the Play section and in the definition of other rules in the Rule section, and are referred to by just their identifiers. Examples of Rule Section syntax follows:

The Rule section should follow the following syntax:

```
Rules { /* rule-statements go here */ }
```

Each rule statements’ syntax is as follows:

```
rule rule-name: /* statements */;

e.g. rule R1:  return true;;

     rule R2:  return R1;;
```

3.5.F Play Section

The Play section of a BoredGames program is meant to outline the progression of a turn. Output and Input functions are used to give the user current information about the Board and input a user's move respectively. BoredGames keeps track of the current player through the NextPlayer function, which can be called at any time and gives a way to alter the turn sequence. The original Player order is the order in which they were defined in the Setup section. Game is ended when the EndGame function is called. EndGame takes a string as a parameter and displays the string in stdout. At the end of Play, the entire block is restarted from the beginning keeping intact the game state (Board, Pieces and Player) and the CurrentPlayer. The syntax for the Play section is as follows:

```
Play { /*statements*/ }
```

3.6 Library Functions

There are a few library functions available to the user. These functions exist to make game actions, such as adding or moving a piece as simple as possible.

Output takes a string as an argument and displays it to the user in the terminal

```
Output(string);
```

Input takes a variable as its argument, gets command line input from the user and stores that input in the argument variable.

```
Input(input_location);
```

move takes a Piece, a x location, and a y location and takes that specific Piece on the Board and moves it to the coordinate given. If multiple Pieces fit the Piece description the first one reached will be moved.

```
move(Piece, x, y);
```

add takes the specified Piece, which is currently in the current Player's inventory and places it on the Board at the x and y coordinates given. If multiple Pieces fit the Piece description the first one reached will be added.

```
add(Piece, x, y);
```

EndGame prints the message supplied in the string and stops the repeated calls to Play. It ends the program entirely.

```
EndGame(string);
```

CurrentPlayer returns the Player whose turn it currently is. To move to the next Player in the list use the NextPlayer function (see below).

```
      CurrentPlayer();
```

NextPlayer moves the return of CurrentPlayer to the next Player in order of Player declarations in the Setup section. Note NextPlayer does not have parentheses following it.

```
      NextPlayer;
```

4. Project Plan

4.1 Plan, Responsibilities and Development

We planned our project through in person meetings throughout the class session. We met at least three times a week to discuss our current statues and to resolve discrepancies in how we thought our language should work. Initially we focused on the proposal splitting up the work evenly between the two of us after finding out that our third member had dropped the class. The LRM also followed this plan. Once we got to coding we split our work up by sections. Brandon handled the Scanner, Sast, and the Semantics checking while Kristen worked on the Parser, Ast, and Code Generation. We assisted each other when problems arose and individually made slight changes to the other's sections in order to correct problems and add new features. We did not follow a specific timeline for our project. We simply worked on each section and completed it as quickly as we could. Finally, this Final Report was written by Brandon.

4.2 Testing

We developed a series of test files toward the end of our project once every component up to and including the Semantics checking was done (leaving just the Code Generation left) in order to make sure all of our features were working correctly. We had an automated test script that would check the output of our tests to make sure everything was running smoothly.

4.3 Development Environment

We used GitHub as our version control system. We avoided branching, to maintain a coherent project, so it was essentially a version control. In addition to this we used Java as the language that BoredGames which was written in OCaml compiles down into. Our environment also utilized shell scripts, Makefile, Ocamllex and Ocamlyacc to compile all of our files.

4.4 Git Log

```
commit 8a914cf973feadf2fb51c5be2f65a83901106f4f
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
```

```
Date:   Fri Aug 15 22:51:25 2014 -0400

    Edited tests

commit 30579ae93c2cf253f098c2dde9facb405640285f
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 21:43:17 2014 -0400

    Edited codeine Added tests

commit e4535b9e06bb6a1188906524b59eac85c967d4de
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 20:37:16 2014 -0400

    Make clean

commit 16db97d1062a358b3aad6f84a12f6443871961c7
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 16:35:30 2014 -0400

    make file, run script and compiler update

commit 6f5f2d6dfce0ad6d1f3cd49a930db4de14f018fd
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 16:27:18 2014 -0400

    more tictac

commit 786cf58d8163dad2c7cef099f9a47acda6812aae
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 16:23:42 2014 -0400

    fixed tictac

commit 5d59661658464e149a3861e8dd9cf71fbf0f5383
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 16:22:15 2014 -0400

    Edited pattern

commit 39c622ad58e05c01e079beba6146c30f0565b3d8
Merge: b1e67dd 8570d51
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 16:20:19 2014 -0400

    Merge branch 'master' of https://github.com/brandonKessler/BOREDGAMES

commit b1e67dd7a6ec027bba44a5fa42d68d6561390861
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 16:20:16 2014 -0400

    update tictactoe

commit 8570d51577af8964e65ad21150489ba0207cb039
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 16:18:19 2014 -0400

    Removed test errors

commit 8f1d1af60e048f9fb093a7302fc4fcfa7f0878d1
Merge: 71e869d b9b82a1
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 16:16:16 2014 -0400

    Merge branch 'master' of https://github.com/brandonKessler/BOREDGAMES

commit 71e869db414def24443b003b52ac664540d27c82
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 16:16:13 2014 -0400
```

```
    new tictactoe

commit b9b82a1d24129a7e9e5e2e6799824300eb8616a0
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 16:14:38 2014 -0400

    Edited codegen for java

commit 270cb8809e239d83f481880204c62d1cd6b6ad66
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 16:07:42 2014 -0400

    Edited tic

commit 9904e7d86c0d233f489551a5b01c7e8769774b5f
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 15:24:41 2014 -0400

    Added java

commit 0e912d9153163b0bfee8427be193f07c8ee7739e
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 14:55:40 2014 -0400

    fixed tictactoe

commit 121d6cee1627677026adf60506cc5adebb06315f
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 14:52:17 2014 -0400

    Edited tictactoe

    Edited deq

commit 1c366ef97f2042c705e009df16a66596ff9cbb4b
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 14:46:23 2014 -0400

    Edited for deq

commit e8a799e6d16cd6fa75104e722ab9dca81126b63b
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 14:38:25 2014 -0400

    Edited for Dequals

commit d33520e97b52f18dc178fd67db46a40e70aa333a
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 14:33:58 2014 -0400

    added dequal

commit 6311373529f70a4868f3fcbb61cc0c894a4ed0c4
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 14:13:40 2014 -0400

    tictactoe

commit 16e340c0e1b1d3b6f51e34f838755e6bab7fc1aa
Author: Brandon <bpk2107@columbia.edu>
Date:   Fri Aug 15 14:12:03 2014 -0400

    tictactoe update

commit 37a8bde55e70fa441bddc7fe7846ce23fa991a19
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 14:07:36 2014 -0400
```

```
    Edited codegen

    Fixed some prints

commit 97b9e17a1c3471e9b533f7c224c2c871f8e6da5e
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 14:07:22 2014 -0400

    Edited tests

commit 5f48d298f23eef03631e3e853299be68f7bf8b0d
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 10:56:30 2014 -0400

    Edited codegen

    removed extra semicolon at add() call

commit 4587addab68804b8504808219ad7a4dcd2050e05
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 10:54:23 2014 -0400

    Edited add so it works, but the add doesn't work

commit 399c0155c8f01305bc24592c326c78cdbf975d44
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 10:45:22 2014 -0400

    Edited loop so it runs

    unoccupied() works

commit a56ebe7dbdbbdfe6deb7f8b4cad00dab314b9763
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 10:40:34 2014 -0400

    Added some complicated loop and add board tests

commit 4cd98b1cd4b09ec69aaa4f94c4baee57d9b8dcaf
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 10:37:55 2014 -0400

    Edited codegen to align versions of array

    When player array is given, I subtract 1 when accessing the Players
    list to account for offset

commit fc2d415855ab2fc213f68f4493ad75d6a8e4e5b2
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 03:55:37 2014 -0400

    Added string and board test

commit f3b0e1e3d751a6ba27f99f4702a7a746a938975e
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 03:24:04 2014 -0400

    Edited codegen and parser

    Changed precedence and added triple DOT

commit 5bee976597e5794ed6058fb71883829aaa87849b
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 02:00:26 2014 -0400

    Edited tests/test.board to separate declaration

commit 318b2b4361eb0f7d80aba39a18682d3605c41416
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
```

```
Date:   Fri Aug 15 01:59:58 2014 -0400

    Edited parser for DOT precedence

commit dbafd7e26592e9f5714a5bacbfa983fd6b16c19c
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Fri Aug 15 01:59:40 2014 -0400

    Edited codegen for testing DOT purposes

commit 5923a40f0232f5f69adc4587fa5266de02687f47
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 23:22:53 2014 -0400

    Edited codegen

    Fixed loop in Pieces create and changed rule return type to bool

commit 7e2259c05932148fd209f87c0823439440d7c292
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 23:12:54 2014 -0400

    Edited codegen

    Changed Loop

commit 947248fff82c69b5ce57f63f55a873ec58ffbae2
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 23:09:52 2014 -0400

    Edited codegen and changed BigTest

commit bb6b152b100066ecc935d525cac59798c1927951
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 22:52:21 2014 -0400

    fixed precedence for cat and changes test files

commit 7190e8e28983e5c35c3ea967e6d4add7e5098662
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 21:05:19 2014 -0400

    CODEGEN WORKS!

commit b61c5001901644c0a29f5468bef408ab781027ca
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 18:48:53 2014 -0400

    fixed shift reduce

commit 7d5450c2541fe17af4d30f96bfb6c67ce152fa50
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 18:37:52 2014 -0400

    Edited parser dot precedence

commit d63d2b32f4f6c637b342c4e8c63bad4bb7e0f32c
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 18:27:29 2014 -0400

    Codegen compiles!!

commit af032b8656be8d0e4a1a24d491414ef8fda62ef2
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 18:16:28 2014 -0400

    Edited codegen

commit 33d605dbb03ab98b5e3d0bf3dd698c3ca8fe93fe
```

```
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 18:16:18 2014 -0400

    Edited east and semantics and compiler

    To work with program triple

commit 4cf57bff0a1deb7998e4bd50a7c7fab662d0b3de
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 18:06:12 2014 -0400

    Edited codegen

    Still have shift/reduce

commit 3e9b3b88b9e0786d4e17cb2cf775259bd7947861
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 17:42:42 2014 -0400

    Revert "updated tests"

    This reverts commit fa7d75458940c52f9ae682d788a6ea186aef2fe1.

commit fa7d75458940c52f9ae682d788a6ea186aef2fe1
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 17:42:31 2014 -0400

    updated tests

commit fe4afd6cfceea9c978b476977b826f5bf4b759e0
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 17:24:43 2014 -0400

    Edited codegen

    Working on fixing

commit a1bd42fb4b217dc7472a2e0cec514f41e97530eb
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 16:52:22 2014 -0400

    cleaned up some unused code

commit 8e81573eecbd0c4ab7ba5f274b736ff0dcc16f7d
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 16:32:04 2014 -0400

    fixed printing in ast

commit 090cfcb876b9fb4f964d44b5df59e53450c4211d
Merge: 17dacac ec8849c
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 16:28:52 2014 -0400

    Merge branch 'master' of https://github.com/brandonKessler/BOREDGAMES

commit 17dacac0841913c155b22779465d864188248cc2
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 16:27:31 2014 -0400

    added cat and fixed board to accept vars

commit ec8849c35a8c5112ade43b1a48515ae8cb928aa0
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 16:00:58 2014 -0400

    Edited codegen comp.sh and compiler

    Partially fixed codegen
```

```
commit 8cc00e7a0a23815c4bf7b22a3e81020031ec303d
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 15:26:52 2014 -0400

    updated tests

commit fee6fde69f9226ad8e5f2d3b2359e7852c6429f3
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 14:17:45 2014 -0400

    update makefile

commit 8cff5706471c2aa47a291c966b7293e30c3772c8
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 14:16:33 2014 -0400

    added makefile

commit fd8a32877d024f2348aaf919f0264091b7e2e82e
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 14:01:25 2014 -0400

    got rid of extra code in semantics

commit 4855f969708c3f2da1b491a38e5f6e181aad6fee
Author: Brandon <bpk2107@columbia.edu>
Date:   Thu Aug 14 13:33:40 2014 -0400

    testing stuff

commit ce95a1d2d6aca4e95ed1a155b80f4cfed703e4ed
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Thu Aug 14 12:46:50 2014 -0400

    Added codegen

commit b16601cfc3d500ac82e2f4e59ec449c4a360d600
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Wed Aug 13 20:31:01 2014 -0400

    Edited cln.sh and comp.sh to include compiler.ml

    Added simple test.txt

commit 0808b00d4654342e2df30e37b0a0c97f9e474149
Author: Brandon <bpk2107@columbia.edu>
Date:   Wed Aug 13 15:13:44 2014 -0400

    Some bug fixes for sast

commit 7a42d209b584d47bbc10b3669b4b1ac0d0c72455
Author: Brandon <bpk2107@columbia.edu>
Date:   Wed Aug 13 14:59:26 2014 -0400

    Somewhat working semantics checking

    semantics checking is working in early tests should need to fix some
    errors and clean up code

commit f17a5b9ce1b0051f4781336830da9663e1735d1d
Author: Brandon <bpk2107@columbia.edu>
Date:   Sat Aug 9 13:37:51 2014 -0400

    changed to new scanning

commit 5a8c5505cc426402409b24a65d299fabe62303a4
Author: Brandon <bpk2107@columbia.edu>
Date:   Sat Aug 9 12:54:10 2014 -0400
```

```
    Added Compiler to test program

commit 1e82a35ff118e6521acea2d6af677e43ddb5d78b
Author: KristenWise <kristenwise13@dyn-160-39-135-153.dyn.columbia.edu>
Date:   Fri Aug 8 17:15:40 2014 -0400

    Edited ast

    Fixed print statement

commit 54cb39414d230426d67640b780afbf177fe5dc96
Author: KristenWise <kristenwise13@dyn-160-39-135-153.dyn.columbia.edu>
Date:   Thu Aug 7 18:47:27 2014 -0400

    Edited ast

    Added print section for ast

commit 3a1c6ac640271afeef14f8f63f48426a473a4d09
Author: KristenWise <kristenwise13@dyn-160-39-135-153.dyn.columbia.edu>
Date:   Thu Aug 7 16:38:02 2014 -0400

    New parser, ast, scanner, and clean/compile shells

commit 91c415f47c6327126911340d2083f9626407cd1b
Author: KristenWise <kristenwise13@dyn-160-39-135-153.dyn.columbia.edu>
Date:   Thu Aug 7 13:19:38 2014 -0400

    Deleted unnecessary parser and ast copies

commit 76dec3c8d865ffa26e8aa74c3a57cd292b256301
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Wed Aug 6 12:22:46 2014 -0400

    Added copy of parser2 as parser3

    Removed extra type variable declarations from parser2

commit ca1f1648d73d7ed1bc07501438032f7693316dbc
Author: Brandon <bpk2107@columbia.edu>
Date:   Wed Aug 6 12:16:08 2014 -0400

    fixed a typo

commit 6012bf9ae297e04cf76a2241995853df149089a2
Author: Brandon <bpk2107@columbia.edu>
Date:   Wed Aug 6 12:14:26 2014 -0400

    changes decls in parser

commit 0efe07fbaf2763c3e374497868d7f641bbe31634
Author: Brandon <bpk2107@columbia.edu>
Date:   Wed Aug 6 10:55:41 2014 -0400

    Changed AST, parser and Scanner so we can compile

commit 6838cf2da35b18c60f41f3ad15e58d5042e98d2f
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Wed Aug 6 10:36:50 2014 -0400

    Edited parser2 and ast2

    Tried to change bg_t so we can cast.

commit 0023b2c4628c1f7c4dfaa875f331a0b2b2c2ffd2
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Wed Aug 6 10:18:06 2014 -0400
```

```
    Added parser2 and ast2

    Reworked the flow of the program; uses statement lists as program
    information.

commit 70df920e8e2eda8b2a2750b3da70bc24a83fcbde
Author: KristenWise <kristenwise13@dyn-160-39-134-244.dyn.columbia.edu>
Date:   Tue Aug 5 18:24:55 2014 -0400

    Edited parser and ast

    Changed stmt setup; still working on it. Might have idea for var
    declarations.

commit 0205e271e665e63a358136bd99e7ddc7539ca912
Author: KristenWise <kristenwise13@dyn-160-39-134-163.dyn.columbia.edu>
Date:   Mon Aug 4 18:04:39 2014 -0400

    Class changes ast and parser

    Changed variable type structure

commit 896013d29ffd5b96d460b8b5fdd38636cbcb3595
Author: KristenWise <kristenwise13@dyn-160-39-134-163.dyn.columbia.edu>
Date:   Mon Aug 4 15:51:14 2014 -0400

    Edited ast and parser

    Changed variable name/type by adding a type vardec_t

commit 00d48c2326c9ff81624bfc6c7012705bfec703d5
Author: Brandon <bpk2107@columbia.edu>
Date:   Mon Aug 4 13:30:08 2014 -0400

    Meeting Changes

    Changes to ast, scanner and parser

commit 1da1928e74e08f54df989fe7a91acb26c4cf0c01
Author: KristenWise <kristenwise13@dyn-160-39-134-163.dyn.columbia.edu>
Date:   Mon Aug 4 09:38:02 2014 -0400

    Added sast

    First part of sast

commit 4ffb0434b2d2b18324eb15570b1cc84625550f87
Author: KristenWise <kristenwise13@dyn-160-39-134-163.dyn.columbia.edu>
Date:   Sun Aug 3 13:02:43 2014 -0400

    Edited parser, ast

    Added ++ and - -

commit f2448f285d8b2cc79b5497297746b0130fdc30e5
Author: KristenWise <kristenwise13@dyn-160-39-134-163.dyn.columbia.edu>
Date:   Sun Aug 3 12:53:46 2014 -0400

    Edited parser

    Changed types. Variable declarations are now stored in tuples,
    (variable name, variable type).

commit 9a9949ad31a721ad32a95335a60e7763fa1afc2f
Author: KristenWise <kristenwise13@dyn-160-39-134-163.dyn.columbia.edu>
Date:   Sun Aug 3 01:57:10 2014 -0400

    Edited parser and ast
```

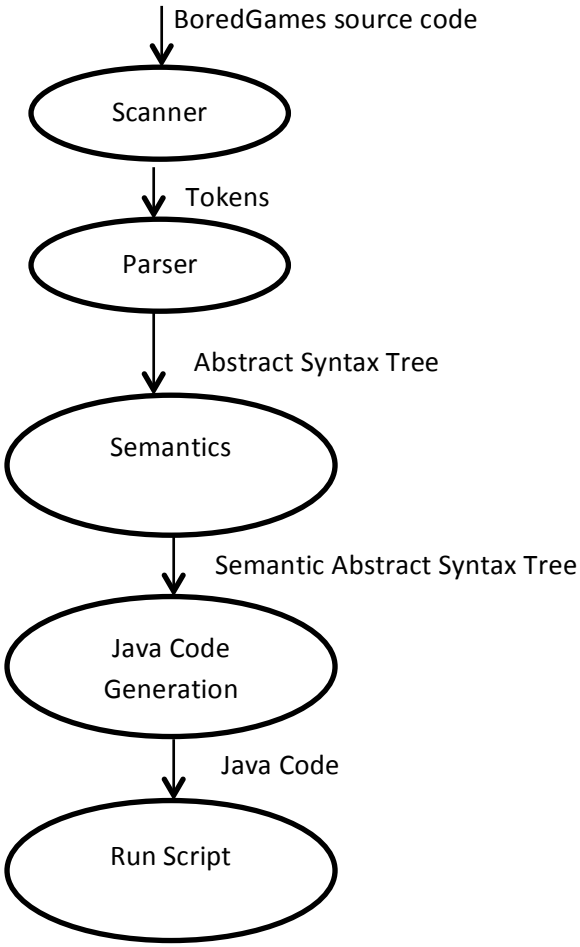
```
    Changed part of dot access. Still uncertain.

commit 5840fda4a31642b008a67f8d23fae930ffb1538d
Author: KristenWise <kristenwise13@dyn-160-39-134-163.dyn.columbia.edu>
Date:   Sat Aug 2 17:55:50 2014 -0400

    Edited parser and ast

    Added colon operator

commit 12200fa4dd66ae14e8ccdb5e5edb64066a657f81
Author: KristenWise <kristenwise13@dyn-160-39-134-163.dyn.columbia.edu>
Date:   Sat Aug 2 12:44:13 2014 -0400

    Edited Parse

    Trying to order the access method for multiple dots ending in array or
    function access.

commit 6922fae641e3600528864d0f11dbad98c22dca4f
Author: KristenWise <kristenwise13@dyn-160-39-134-163.dyn.columbia.edu>
Date:   Sat Aug 2 11:09:05 2014 -0400

    Edited ast

    Changed some of the types and organization (double to float, added bg_t
    to hold each individual type). Trying to figure what goes in ast vs
    sast.

commit 68eb5341708167e80eb9fd1bc12e625e602963f5
Author: KristenWise <kristenwise13@dyn-160-39-134-163.dyn.columbia.edu>
Date:   Fri Aug 1 22:09:04 2014 -0400

    Edited parser and ast

    Changed program organization(parser and sat); removed game type, made
    program type larger (parser and ast); changed method of return from
    each section (parser)

commit 0915a2025a518cd9fad6be082a71f7bfb231b218
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Wed Jul 30 17:18:21 2014 -0400

    Edited ast

    Added section for types

commit d5ee27baacc261feea7346a62ebeeabb7756d061
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Wed Jul 30 17:17:49 2014 -0400

    Edited parser

    Changed types, removed parenthesis

commit 6abe0d86bd6a548315b1bddc76749db26cfd8858
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Wed Jul 30 13:27:30 2014 -0400

    Edited parser

    Added access for array using int, though not yet for matrix using
    coordinate

commit 2959100035d84d5b483ee0b81d7c0fa54b1375d0
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Wed Jul 30 13:21:59 2014 -0400

    Edited ast
```

```
    Added “Access” under expression, to access arrays

commit 31ba61400aabde94a4865aed4b24e2e9af2e1e35
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Wed Jul 30 13:01:47 2014 -0400

    Edited parser

    Added part for Play, added keywords for the brackets

commit 4dcb35c61a95e7f045cfee9f27860024ac906437
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Wed Jul 30 12:55:49 2014 -0400

    Edited ast

    Added locals tuple to rules for variables declared there

commit 9d6c447ecadf855f30a54171ae58eb69297ef436
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Tue Jul 29 18:26:29 2014 -0400

    Edited Parser

    Added rules section

commit 1ee6faa708daf9f0e07dac950ecb1f83a2351190
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Tue Jul 29 18:08:16 2014 -0400

    Edited ast

    Added rules_t

commit 3a5d3a765be83d4c2e58d0161c981abf8237b441
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Tue Jul 29 17:54:00 2014 -0400

    Edited parser

    Finished setup, though it is likely unfinished

    Questions about: using coord_t vs tuples, interleaving different types
    of declarations; using actual integer values, like 0

commit fbaf96a3a923c3559c00339396d21c27f2b81252
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Tue Jul 29 17:21:05 2014 -0400

    Edited ast

    Changed program type from string list * game_t list to a tuple of
    strings list * game_t list

commit c51faa27e8bde7fc0f3a43dff03ce2713844774d
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Tue Jul 29 17:09:33 2014 -0400

    Edited types in ast

    Added game_t to be returned by program; Added coord_t; Added coordinate
    to piece_t

commit 953610c94dcfb72f84fe3b292ac905f97c117d4c
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Tue Jul 29 16:58:30 2014 -0400

    Added ast file
```

```
    First part of it; trying to get type signatures figured out.
    Program returns a list of all variables, including Players, and list of
    Pieces.

commit aed15037881186cb2e5b2cde973b996485046328
Author: Brandon <bpk2107@columbia.edu>
Date:   Wed Jul 30 11:05:32 2014 -0400

    Added Scanner

    Scanner should be working. Need to check string literals for correctness

commit 8db1f4221bab56f6d1427d8bfe289fd3ec2e83ba
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Tue Jul 29 11:25:11 2014 -0400

    Edited parser's program structure

    Added parts for each section; about to add parts for each item

commit c53e1f3040e724e481f4a693206c125c559d6708
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Tue Jul 29 10:37:42 2014 -0400

    Added parser

    First part of parser; the stuff before actual parsing

commit 8215a3992742fa8a95fab328c7ae7f615b8f84fe
Author: Brandon <bpk2107@columbia.edu>
Date:   Tue Jul 29 10:24:05 2014 -0400

    deleted tests

commit ff29a54c2a532cf5896e99d054b628f49a69527b
Author: KristenWise <kristenwise13@dyn-160-39-135-148.dyn.columbia.edu>
Date:   Tue Jul 29 10:18:16 2014 -0400

    TEST

commit 05871ad8c25187da362b2df184e35caa6a322066
Author: Brandon <bpk2107@columbia.edu>
Date:   Tue Jul 29 10:16:51 2014 -0400

    test

commit 46013daa427463e77978f7cbf7d09bb19216f809
Author: Brandon <bpk2107@columbia.edu>
Date:   Mon Jul 28 13:13:44 2014 -0400

    deleted tests

commit 767e64e5c02435d48ceb1d80699c465dd8389cee
Author: Brandon <bpk2107@columbia.edu>
Date:   Mon Jul 28 13:10:50 2014 -0400

    a test

commit 55274205e72e0b75f1f9f27751d97884ce0a736f
Author: brandonKessler <bpk2107@columbia.edu>
Date:   Mon Jul 28 13:11:07 2014 -0400

    test2

commit 26d0dc03d8f00172c3e2bbb0d39ee627c00694a7
Author: Brandon <bpk2107@columbia.edu>
Date:   Mon Jul 28 13:08:22 2014 -0400
```

5. Architectural Design

The following is a diagram representing the architecture of BoredGames



5.1 Scanner

The scanner tokenizes the input BoredGames source code. Whitespaces and comments are discarded and illegal combinations of characters are caught here.

5.2 Parser

The Parser takes the tokens from the Scanner and uses these tokens to generate the Abstract Syntax Tree. Syntax errors are caught in this section.

5.3 Semantics

The Semantics checking section takes the Abstract Syntax Tree generated by the parser and walks through it doing semantical checking and produces a typed Semantic Abstract Syntax

Tree. Variables scope is taken into account in this section when the semantic checking is occurring. In addition to type checking the Semantics Checking also makes sure that arguments of operators are valid. An error message is generated if the Semantic Checker finds anything wrong and once the message is generated all compilation stops.

5.4 Java Code Generation

In the Code Generation step we walk through the Semantic Abstract Syntax Tree and generate java code that corresponds to the tree. All of the code is generated to a single .java file.

5.5 Run Script

The Run Script takes our java code generated by the Java Code Generation and compiles it along with the prebuilt java classes to create an executable file.

6. Test Plan

Our testing comprised of generating java code for a large group of files and testing the output of each individually.

File Name	Description Of Test
access.board	Tests various piece/board accesses
add.board	Tests adding to the board
BigTest.board	Hits almost all pieces of our language
binop.board	Tests binary operations
current.board	Tests various current player and accesses
dsa.board	Tests rules and strings
elseiftest.board	Tests if and else if statements
greaterthantest.board	Tests the greater than operator
helloworld.board	Displays hello world
ifelsetest.board	Tests if and else statements
loop.board	Tests loops
piecetest.board	Tests piece creation
players.board	Tests various player configurations
playertest.board	Tests player creation
rule.board	Tests rule use
ruleCheck.board	Tests rule use further
rules.board	Test multiple rules
stringbd.board	Tests access on board
test.board	Tests EndGame
tictactoe.board	Plays TicTacToe
wrongtypecast.board	Generates a wrong typecast error

Some of these tests have depreciated due to considerable changes in our program. Non-depreciated tests are part of our testing automation. Test cases can be found in our tests folder in our source code submission. helloworld.board and tictactoe.board are included in the Complete Listing section for convenience.

6.1 Automated Testing

After compiling using Make, running ./tests.sh will run all non-depreciated tests and display the output of the program. The final test is tic-tac-toe, which requires user input to play. The rest of the tests should text as output.

7. Lessons Learned

Brandon Kessler

The most important think I learned working on this project was to manage time efficiently. Wherever you think you should be at a certain date you should be farther. This leaves room for problems you will run into toward the end of the project. Also make sure to communicate changes made in how things should work in your language effectively. It sucks to write code and then realize you have to change it because you didn't tell your partner that you added it. Overall, my biggest advice would be to not be afraid to drop content from your language if it become overwhelming.

Kristen Wise

Communication is key. Talking about confusing code is hard enough, but adding another layer of indirection does not work as well with people as it does compilers (<-joke, but still true). Meeting in person makes it about 20 times easier to understanding everyone's code than it does over email or text. Also, debugging as a group, even if everyone is doing his/her own thing, facilitates collaboration; it is likely that if one person never seen a given error, someone else has. Planning the interfaces between program sections is crucial; misunderstandings can require massive overhauls. Sometimes you want to look under the hood of ocaml and see how things are being processed, but all you have is your and your partner(s) code. My biggest piece of advice is to meet and meet often. It is easy to get behind, or think that you are further than you actually are, and meeting in person helps keep things on track.

8. Complete Listing

scanner.mll

```
{ open Parser }


rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LBRACKET }
| ']'      { RBRACKET }
| ';'      { SEMI }
| ','      { COMMA }
| ':'      { COLON }
| '.'      { DOT }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '^'      { CAT }
| "++"     { PLUSPLUS }
| "--"     { MINUSMINUS }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
```

```
      | ">"      { GT }
      | ">="     { GEQ }
      (* add not?*)
      | ".=" { DEQ }
      | "&&"     { AND }
      | "||"     { OR }
      | "if"     { IF }
      | "else"   { ELSE }
      | "loop"   { LOOP }
      | "return" { RETURN }
      | "int"    { INT }
      | "bool"   { BOOL }
      | "string" { STRING }
      | "double" { FLOAT }
      | "piece"  { PIECE }
      | "new Board"  { BOARD }
      | "new Player" { PLAYER }
      | "new Pieces" { PIECES }
      | "Setup"  { SETUP }
      | "Rules"  { RULES }
      | "rule"   { RULE }
      | "Play"   { PLAY }
      | "NextPlayer" { NEXTPLAYER }
      | "Dice" { DICE }
      | ('-')? ['0'-'9']+ as lxm { INTLITERAL(int_of_string lxm) }
      | ("true" | "false") as lxm { BOOLLITERAL(bool_of_string lxm) }
      | ('-')? (['0'-'9'])+ '.' (['0' - '9'])* as lxm { FLOATLITERAL(float_of_string
lxm) }
      | '"' [^'"']* '"'  as lxm  { STRINGLITERAL(lxm) } (* check for correctness *)
      | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
```

```
      | eof { EOF }
      | _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

      and comment = parse
        "*/" { token lexbuf }
      | _    { comment lexbuf }
```

parser.mly

```
      %{ open Ast %}

      %token SEMI LBRACKET RBRACKET LPAREN RPAREN LBRACE RBRACE COMMA
      %token PLUS MINUS TIMES DIVIDE COLON DOT PLUSPLUS MINUSMINUS
      %token ASSIGN EQ NEQ LT LEQ GT GEQ RETURN IF ELSE LOOP EOF AND OR CAT DEQ
      %token INT BOOL FLOAT STRING BOARD RULE PIECES PLAYER PIECE NEXTPLAYER DICE
      %token SETUP RULES PLAY
      %token <int> INTLITERAL
      %token <string> ID
      %token <bool> BOOLLITERAL
      %token <float> FLOATLITERAL
      %token <string> STRINGLITERAL



      %nonassoc NOELSE
      %nonassoc ELSE
      %right ASSIGN
      %left AND OR
      %left EQ NEQ
```

```
%left LT GT LEQ GEQ DEQ

%left CAT

%left PLUS MINUS

%left TIMES DIVIDE

%left MINUSMINUS PLUSPLUS

%nonassoc COLON /* is this right */

%left DOT

%nonassoc LPAREN RPAREN RBRACKET LBRACKET



%start program

%type <Ast.program> program


%%


program:

   setup rules play { ($1,$2,$3) }


setup:

   SETUP LBRACE setup_list RBRACE      { List.rev $3 }


rules:

   RULES LBRACE rule_list RBRACE { List.rev $3 }


play:

   PLAY LBRACE stmt_list RBRACE        { List.rev $3 }


var_dec:
```

```
   vdecl SEMI       { $1 }


vdecl:
   INT var          { (Int,$2) }
 | FLOAT var        { (Float,$2) }
 | STRING var       { (String,$2) }
 | BOOL      var          { (Bool,$2) }
 | PIECE var            { (Piece,$2) }


var:
   ID               { Id($1) }
 | ID ASSIGN expr   { Assign($1,$3) }


setup_list:
   setup_dec              { [$1] }
 | setup_list setup_dec          { $2::$1 }


setup_dec:
   bdecl            { Setbd($1) }
 | pldecl           { Setplr($1) }
 | pcdecl           { Setpc($1) }
 | stmt                   { Stmt($1) }



pldecl:
   PLAYER LPAREN STRINGLITERAL RPAREN SEMI{ {plrname = $3} }


pcdecl:
```

```
   PIECES LPAREN pcargs RPAREN SEMI    { $3 }



pcargs:
   STRINGLITERAL COMMA STRINGLITERAL COMMA INTLITERAL
             { {owner = $1; name = $3; num = $5; ptval = 0; cloc =
                       {xc=Lint(0); yc=Lint(0)}} }
  | STRINGLITERAL COMMA STRINGLITERAL COMMA INTLITERAL COMMA INTLITERAL
             { {owner = $1; name = $3; num = $5; ptval = $7; cloc =
                       {xc=Lint(0); yc=Lint(0)}} }



bdecl:
   BOARD LPAREN INTLITERAL COMMA INTLITERAL RPAREN SEMI    { {rows = $3; cols
= $5} }



rule_list:
   /* nothing */          { [] }
  | rule_list rule_dec            { $2 :: $1 }


rule_dec:
   RULE ID COLON stmt_list SEMI
             { {rname= $2; rbody = List.rev $4} }



stmt_list:
   /* nothing */    { [] }
```

```
 | stmt_list stmt   { $2 :: $1 }


stmt:
   expr SEMI              { Expr($1) }
 | RETURN expr SEMI       { Return($2) }
 | LBRACE stmt_list RBRACE{ Block(List.rev $2) }
 | IF LPAREN expr RPAREN stmt %prec NOELSE    { If($3, $5, Block([])) }
 | IF LPAREN expr RPAREN stmt ELSE stmt       { If($3, $5, $7) }
 | LOOP LPAREN expr RPAREN stmt        { Loop($3, $5) }
 | var_dec                             { Decl(fst $1, snd $1) }
 | NEXTPLAYER SEMI        {NextPlayer}


expr:
   INTLITERAL       { Lint($1) }
 | FLOATLITERAL     { Lfloat($1) }
 | STRINGLITERAL    { Lstring($1) }
 | BOOLLITERAL            { Lbool($1) }
 | ID               { Id($1) }
 | 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 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) }
 | expr GT expr           { Binop($1, Greater, $3) }
 | expr GEQ expr    { Binop($1, Geq, $3) }
```

```
       | expr OR expr           { Binop($1, Or, $3) }
       | expr AND expr    { Binop($1, And, $3) }
       | expr DEQ expr            { Binop($1, Dequal, $3)}
       | expr COLON expr  { Through($1, $3) }
       | expr CAT expr        { Cat ($1, $3) }
       | ID ASSIGN expr   { Assign($1, $3) }
       | expr DOT expr    { Daccess($1, $3) }
       | expr PLUSPLUS    { Incr($1,Plus) }
       | expr MINUSMINUS  { Incr($1,Minus) }
       | expr LPAREN actuals RPAREN          { Call($1, $3) }
       | LPAREN expr RPAREN     { $2 }
       | expr LBRACKET expr RBRACKET   { Access($1, $3) }
       | expr LBRACKET LPAREN expr COMMA expr RPAREN RBRACKET    {
Baccess($1,{xc=$4;yc=$6}) }



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


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

ast.ml

```
      type op = Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
Or | And | Dequal
```

```
type bg_t = Int | Float | Bool| String | Piece | Player | Tile  | Rule


type inc = Plus | Minus




type player_t = {
   plrname : string;
}




type mat_t = {
   rows : int;
   cols : int;
}



type expr =
   Lint of int
 | Lfloat of float
 | Lbool of bool
 | Lstring of string
 | Id of string
 | Binop of expr * op * expr
 | Cat of expr * expr
 | Through of expr * expr
```

```
    | Incr of expr * inc
    | Assign of string * expr
    | Call of expr * expr list
    | Access of expr * expr
    | Baccess of expr * coord_t
    | Daccess of expr * expr
    | Noexpr
and coord_t = {
    xc : expr;
    yc : expr;
}
type piece_t = {
    owner : string;
    name : string;
    num : int;
    ptval : int;
    cloc : coord_t;
}
type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | Loop of expr * stmt
  | Decl of bg_t * expr
  | NextPlayer
type setup_dec =
    Setbd of mat_t
```

```
 | Setpc of piece_t
 | Setplr of player_t
 | Stmt of stmt

type rules_t = {
   rname : string;
   rbody : stmt list;
 }

type program = setup_dec list * rules_t list * stmt list



let rec string_of_expr = function
   Lint(l) -> string_of_int l
 | Lfloat(f) -> string_of_float f
 | Lbool(b) -> string_of_bool b
 | Lstring(st) -> st
 | Id(s) -> s
 | Binop(e1, o, e2) ->
       string_of_expr e1 ^ " " ^
       (match o with
         Add -> "+" | Sub -> "-" | Mult -> "*" | Div -> "/"
       | Equal -> "==" | Neq -> "!="
         | Less -> "<" | Leq -> "<=" | Greater -> ">" | Geq -> ">="
       | Or -> "||" | And -> "&&" | Dequal -> ".equal()") ^ " " ^
      string_of_expr e2
 | Cat(e1, e2) -> string_of_expr e1 ^ " ^ " ^ string_of_expr e2
 | Through(e1,e2) ->
```

```
            string_of_expr e1 ^ ":" ^ string_of_expr e2
      | Incr(e,i) ->
            string_of_expr e ^
            (match i with
              Plus -> "++" | Minus -> "--")
      | Assign(v,e) -> v ^ " = " ^ string_of_expr e
      | Call(f,el) ->
            string_of_expr f ^ "(" ^ String.concat ", " (List.map string_of_expr el)
^ ")"
      | Access(e1,e2) ->
            string_of_expr e1 ^ "[" ^ string_of_expr e2 ^ "]"
      | Baccess(e,c) ->
            string_of_expr e ^ "[(" ^ string_of_expr c.xc ^ ","
            ^ string_of_expr c.yc ^ ")]"
      | Daccess(e1,e2) ->
            string_of_expr e1 ^ "." ^ string_of_expr e2
      | Noexpr -> ""



     let rec string_of_stmt = function
        Block(stmts) ->
            "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
      | Expr(expr) -> string_of_expr expr ^ ";\n";
      | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
      | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
      | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
           string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
      | Loop(e, s) -> "loop (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
```

```
        | Decl(bg,e) ->
              (match bg with
                Int -> "int" | Float -> "float" | Bool -> "bool"
              | String -> "string"
              | Piece -> "piece" |  Player -> "player" | Tile ->
                              "tile" | Rule -> "rule") ^ " " ^
              string_of_expr e
        | NextPlayer -> "NextPlayer"
      let string_of_setup = function
          Setbd(m) -> "new Board(" ^ string_of_int m.rows ^ "," ^ string_of_int
m.cols ^ ")\n"
         | Setpc(pc) -> "new Pieces(" ^ pc.owner ^ ", " ^ pc.name ^ ", " ^
              string_of_int pc.num ^ ", (" ^ string_of_expr pc.cloc.xc ^ "," ^
              string_of_expr pc.cloc.yc ^ "))\n"
        | Setplr(plr) -> "new Player(" ^ plr.plrname ^ ")\n"
        | Stmt(s) -> string_of_stmt s ^ "\n"

      let string_of_rules r = "rule " ^ r.rname ^ ": " ^ String.concat ""
              (List.map string_of_stmt r.rbody) ^ "\n"

      let string_of_program (su,r,st) =
         "Setup {\n" ^ String.concat "" (List.map string_of_setup su) ^ "}\n" ^
         "Rules {\n" ^ String.concat "" (List.map string_of_rules r) ^ "}\n" ^
         "Play {\n" ^ String.concat "\n" (List.map string_of_stmt st) ^ "}\n"
```

sast.ml

```
      open Ast
```

```
type overallTypes = Players | Pieces

type scope = Global

             | Local

type datatype =

     Datatype of Ast.bg_t


type splayer_t = {

   splrname : string;

}


type smat_t = {

   srows : int;

   scols : int;

}

type sexpr =

   SLint of int * datatype

 | SLfloat of float * datatype

 | SLbool of bool * datatype

 | SLstring of string * datatype

 | SId of string * scope * datatype

 | SBinop of sexpr * op * sexpr * datatype

 | SCat of sexpr * sexpr * datatype

 | SThrough of sexpr * sexpr * datatype

 | SIncr of sexpr * Ast.inc * datatype

 | SAssign of string * sexpr * scope * datatype

 | SCall of expr * sexpr list * scope * datatype

 | SAccess of sexpr * sexpr * datatype

 | SBaccess of sexpr * scoord_t * datatype
```

```
    | SDaccess of sexpr * sexpr * datatype
    | SNoexpr
  and scoord_t = {
     sxc : sexpr;
     syc : sexpr;
  }
  type spiece_t = {
     sowner : string;
     sname : string;
     snum : int;
     sptval : int;
     scloc : scoord_t;
  }
   type sstmt =
     SBlock of sstmt list
    | SExpr of sexpr
    | SReturn of sexpr
    | SIf of sexpr * sstmt * sstmt
    | SLoop of sexpr * sstmt
    | SDecl of datatype * sexpr * scope
    | SNextPlayer
   type ssetup_dec =
     SSetbd of smat_t
    | SSetpc of spiece_t
    | SSetplr of splayer_t
    | SStmt of sstmt


   type srules_t = {
```

```
  srname : string;
  srbody : sstmt list;
}
type srules_decl =
        SRules_Decl of srules_t * datatype
type sprogram =
        ssetup_dec list * srules_decl list * sstmt list
```

semantics.ml

```
open Ast
open Sast

exception Error of string

type symbol_table = { parent : symbol_table option; variables : (string *
                        datatype * bg_t option) list;
}

type rules_table = { rules: (string *  sstmt list) list }

type translation_environment = {
        return_type: datatype;
        return_seen: bool;
        location: string;
        global_scope : symbol_table;
        local_scope: symbol_table;
```

```
        rule_scope: rules_table;
}
let rec find_rule (rule_scope : rules_table) name = List.find(fun (s,_) ->
        s=name) rule_scope.rules
let math e1 e2 = match (e1, e2) with
        (Int, Int) -> (Int,true)
        |(Int, Float) -> (Float, true)
        |(Float, Int) -> (Float, true)
        |(Float, Float) -> (Float, true)
        |(_,_) -> (Int, false)
let relations e1 e2 = match(e1, e2) with
        (Int, Int) -> (Bool, true)
        | (Float, Float) -> (Bool,true)
        | (Int, Float) -> (Bool, true)
        | (Float, Int) -> (Bool, true)
        | (_,_) -> (Bool, false)
let logic e1 e2 = match(e1, e2) with
        (Bool, Bool) -> (Bool,true)
        | (_,_) -> (Int, false)
let equality e1 e2 = match(e1, e2) with
        (Bool, Bool) -> (Bool, true)
        | (Int, Int) -> (Bool, true)
        | (Float, Int) -> (Bool, true)
        | (Int, Float) -> (Bool, true)
        | (Float, Float) -> (Bool, true)
        | (String, String) -> (Bool, true)
        | (Piece, Piece) -> (Bool, true)
        | (Player, Player) -> (Bool,true)
```

```
        | (_,_) -> (Int, false)
let catcheck e1 e2 = match (e1, e2) with
        (String, Int) -> (String, true)
        |(String, Float) -> (String, true)
        |(String, String) -> (String, true)
        |(Int, String) -> (String, true)
        |(Float, String) -> (String, true)
        | (_,_) -> (Int, false)
let rec get_type = function
        Datatype(t) -> t


let get_op_return_value ops e1 e2 =
        let type1 = get_type e1 and type2 = get_type e2 in
        let(t,valid) = match ops with
              Add -> math type1 type2
              | Sub -> math type1 type2
              | Mult -> math type1 type2
              | Div -> math type1 type2
              | Equal -> equality type1 type2
              | Neq -> equality type1 type2
              | Less -> relations type1 type2
              | Leq -> relations type1 type2
              | Greater -> relations type1 type2
              | Geq -> relations type1 type2
              | Or -> logic type1 type2
              | And -> logic type1 type2
              | Dequal -> equality type1 type2
```

```
               in (Datatype(t), valid)


       let find_var env name = try List.find(fun (s,_,_) -> s=name)
                env.local_scope.variables with Not_found -> try List.find(fun(s,_,_)
->
                        s=name) env.global_scope.variables with Not_found -> raise
                        (Error("variable not found"))




       let rec check_expr env e = match e with
               Lint(i) -> Datatype(Int)
               | Lfloat(f) -> Datatype(Float)
               | Lbool(b) -> Datatype(Bool)
               | Lstring(s) -> Datatype(String)
               | Id(s) -> let (_,stype, _) = try find_var env s with Not_found ->
                               raise(Error("Undeclared Variable Identifier")) in
stype
               | Cat( e1,e2) -> let type1 = check_expr env e1 and type2 = check_expr
               env e2 in let (t,valid) = catcheck (get_type type1) (get_type type2)
in
                       if valid then Datatype(t) else raise(Error("Bad
Concatination"))
               | Binop(e1, op, e2) -> let type1 = check_expr env e1 and type2 =
                                         check_expr env e2 in
                                       let (t, valid) = get_op_return_value op type1
                                       type2 in
                                       if valid then t else raise(Error("Incompatible
                                        type for operation"));
               |Through(e1, e2) -> let type1 = check_expr env e1 and type2 =
check_expr
```

```
                                            env e2 in
                                            if (type1 = Datatype(Int) && type2 =
                                                    Datatype(Int)) then
                                                            Datatype(Bool)
                                            else raise(Error("Through needs two
ints
                                            as parameters"));
              | Incr(e1, inc) -> let type1 = check_expr env e1 in
                     if type1 = Datatype(Int) then type1 else if type1 =
                             Datatype(Float) then type1 else raise (Error("Cannot
                             perform value change on non numeric datatype"))
              | Assign(s1, e1) -> let (_,type1,_) = (find_var env s1)
                                  and type2 = check_expr env e1 in
                                   (if not (type1 = type2) then (raise (Error("Types
in
                                   variable assignment are not matching"))));
                                   check_expr env e1
              | Access(e1, e2) ->  let typee2 = check_expr env e2 and type1 =
                     check_expr env e1 in
                                      if typee2 = Datatype(Int) then type1 else
raise
                                      (Error("Lookup index needs to be of type
int"))
              | Call(Id("Output"), e1) -> let _ = List.map(fun exp -> check_expr env
                                             exp) e1 in Datatype(Bool)
              | Call(Id("Input"), e1) -> let _ = List.map(fun exp -> check_expr env
exp)
                                            e1 in Datatype(Bool)
              | Call(Id("inventory"), e1) -> let _ = List.map(fun exp -> check_expr
env
```

```
                                            exp) e1 in Datatype(Piece)
              | Call(Id("onBoard"), e1) -> let _ = List.map(fun exp -> check_expr
env
                                             exp) e1 in Datatype(Piece)
              | Call(Id("Pieces"), e1) -> let _ = List.map(fun exp -> check_expr env
                                              exp) e1 in Datatype(Piece)
              | Call(Id("locationx"), e1) -> let _ = List.map(fun exp -> check_expr
env
                                              exp) e1 in Datatype(Int)
              | Call(Id("locationy"), e1) -> let _ = List.map(fun exp -> check_expr
env
                                              exp) e1 in Datatype(Int)
              | Call(Id("name"), e1) -> let _ = List.map(fun exp -> check_expr env
                                              exp) e1 in Datatype(String)
              | Call(Id("point"), e1) -> let _ = List.map(fun exp -> check_expr env
                                              exp) e1 in Datatype(Int)
              | Call(Id("unoccupied"), e1) -> let _ = List.map(fun exp -> check_expr
env
                                              exp) e1 in Datatype(Bool)
              | Call(Id("owner"), e1) -> let _ = List.map(fun exp -> check_expr env
                                              exp) e1 in Datatype(String)
              | Call(Id("move"), e1) -> let _ = List.map(fun exp -> check_expr env
                                              exp) e1 in Datatype(Bool)
              | Call(Id("add"), e1) -> let _ = List.map(fun exp -> check_expr env
                                              exp) e1 in Datatype(Bool)
              | Call(Id("EndGame"), e1) -> let _ = List.map(fun exp -> check_expr
env
                                              exp) e1 in Datatype(String)
              | Call(Id("CurrentPlayer"), e1) -> let _ = List.map(fun exp ->
check_expr env
```

```
                                                exp) e1 in Datatype(Player)
              | Call(id, e1) -> raise(Error("Function does not exist"))


              | Baccess(id, c) -> Datatype(Tile)
              | Daccess(e1, e2) -> let type1 = check_expr env e1 and type2 =
                      check_expr env e2 in
                              if ( type1 = Datatype(Player) || type1 =
                                      Datatype(Piece) || type1 = Datatype(Tile))
then
                                      type2 else raise(Error("Invalid dot access"))
              | Noexpr ->  Datatype(String) (* is this ever used?*)


      let get_var_scope env name = try( let(_,_,_) = List.find (fun (s,_,_) ->
s=name)
              env.local_scope.variables in Local) with Not_found -> try(let(_,_,_) =
List.find
              (fun (s,_,_) -> s=name) env.global_scope.variables in Global) with
Not_found ->
              raise(Error("cant get variable scope"))


      let rec get_sexpr env e = match e with
               Lint(i) -> SLint(i, Datatype(Int))
              | Lfloat(f) -> SLfloat(f, Datatype(Float))
              | Lbool(b) -> SLbool(b, Datatype(Bool))
              | Lstring(s) -> SLstring(s, Datatype(String))
              | Id(s) -> SId(s, get_var_scope env s, check_expr env e)
              | Binop(e1, op, e2) -> SBinop(get_sexpr env e1, op, get_sexpr env e2,
                      check_expr env e)
              | Cat(e1, e2) -> SCat(get_sexpr env e1, get_sexpr env e2, check_expr
env
```

```
              e)
              | Through(e1,e2) -> SThrough(get_sexpr env e1, get_sexpr env e2,
                      check_expr env e1)
              | Incr(e1, inc) -> SIncr(get_sexpr env e1, inc, check_expr env e)
              | Assign(s, e1) -> SAssign(s, get_sexpr env e1, get_var_scope env s,
                       check_expr env e1)
              | Call(Id("Output"), e1) -> let  s_ex_list = List.map(fun exp ->
get_sexpr env exp) e1 in
                                          SCall(Id("Output"),s_ex_list, Global,
check_expr env e)
              | Call(Id("Input"), e1) -> let  s_ex_list = List.map(fun exp ->
get_sexpr env exp) e1 in
                                          SCall(Id("Input"),s_ex_list, Global,
check_expr env e)
              | Call(Id("inventory"), e1) -> let s_ex_list = List.map(fun exp ->
get_sexpr env exp) e1 in
                                          SCall(Id("inventory"),s_ex_list, Global,
check_expr env e)
              | Call(Id("onBoard"), e1) ->  let s_ex_list =  List.map(fun exp ->
get_sexpr env exp) e1 in
                                          SCall(Id("onBoard"),s_ex_list, Global,
check_expr env e)
              | Call(Id("Pieces"), e1) ->  let s_ex_list = List.map(fun exp ->
get_sexpr env exp) e1 in
                                          SCall(Id("Pieces"),s_ex_list, Global,
check_expr env e)
              | Call(Id("locationx"), e1) ->  let s_ex_list =  List.map(fun exp ->
get_sexpr env exp) e1 in
                                          SCall(Id("locationx"),s_ex_list, Global,
check_expr env e)
              | Call(Id("locationy"), e1) ->  let s_ex_list =  List.map(fun exp ->
get_sexpr env exp) e1 in
                                          SCall(Id("locationy"),s_ex_list, Global,
check_expr env e)
```

```
              | Call(Id("name"), e1) ->  let  s_ex_list = List.map(fun exp ->
get_sexpr env exp) e1 in

                                       SCall(Id("name"),s_ex_list, Global,
check_expr env e)

              | Call(Id("point"), e1) -> let  s_ex_list = List.map(fun exp ->
get_sexpr env exp) e1 in

                                       SCall(Id("point"),s_ex_list, Global,
check_expr env e)

              | Call(Id("unoccupied"), e1) ->  let s_ex_list = List.map(fun exp ->
get_sexpr env exp) e1 in

                                       SCall(Id("unoccupied"),s_ex_list, Global,
check_expr env e)

              | Call(Id("owner"), e1) ->  let s_ex_list =  List.map(fun exp ->
get_sexpr env exp) e1 in

                                       SCall(Id("owner"),s_ex_list, Global,
check_expr env e)

              | Call(Id("move"), e1) ->  let s_ex_list =  List.map(fun exp ->
get_sexpr env exp) e1 in

                                       SCall(Id("move"),s_ex_list, Global,
check_expr env e)

              | Call(Id("add"), e1) ->  let s_ex_list = List.map(fun exp ->
get_sexpr env exp) e1 in

                                       SCall(Id("add"),s_ex_list, Global, check_expr
env e)

              | Call(Id("EndGame"), e1) ->  let s_ex_list =  List.map(fun exp ->
get_sexpr env exp) e1 in

                                       SCall(Id("EndGame"),s_ex_list, Global,
check_expr env e)

               | Call(Id("CurrentPlayer"), e1) ->  let s_ex_list =  List.map(fun exp
-> get_sexpr env exp) e1 in

                                       SCall(Id("CurrentPlayer"),s_ex_list, Global,
check_expr env e)

              | Call(id, e1) -> raise(Error("Function does not exist"))
```

```
            | Baccess(id, c) -> SBaccess(get_sexpr env id, {sxc = get_sexpr env
            c.xc; syc= get_sexpr env c.yc}  ,check_expr env e)
            | Daccess(e1, e2) -> SDaccess(get_sexpr env e1, get_sexpr env e2,
            check_expr env e)
            | Noexpr -> SNoexpr
            | Access(e1, e2) -> SAccess(get_sexpr env e1, get_sexpr env e2,
            check_expr env e)

    let get_sexpr_list env expr_list = let sexpr_list= List.map(fun expr -> let t1
    = get_type(check_expr env (List.hd expr_list)) and t2 =
            get_type(check_expr env expr) in if(t1=t2) then get_sexpr
            env expr else raise(Error("Type Mistmatch"))) expr_list in sexpr_list

    let get_sdecl env decl = match decl with
            Decl(datatype, Id(s)) -> SDecl(Datatype(datatype), SId(s, Global,
            Datatype(datatype)), Global)
            |Decl(datatype, Assign(s,e1)) ->  SDecl(Datatype(datatype), SAssign(s,
            get_sexpr env e1, Global, check_expr env e1),Global)
            | _ -> raise(Error("Bad Declaration"))
    let get_name_type_from_decl decl = match decl with
            Decl(datatype, Id(s)) -> (s, datatype)
            | Decl(datatype, Assign(v,e)) -> (v, datatype)
            | _ ->raise(Error("Bad variable declaration"))
    let get_name_type_value_from_decl decl = match decl with
            Decl(datatype, Id(s)) -> (s, datatype, None)
            | Decl(datatype, Assign(v,e)) -> (v, datatype, Some(e))
            | _ ->raise(Error("Bad variable declaration"))
```

```
	let add_to_global_table env name t v =
		 let new_vars = (name,t,v):: env.global_scope.variables in
		let new_sym_table = {parent = env.global_scope.parent; variables =
			new_vars;} in
		let new_env = {env with global_scope = new_sym_table} in
		new_env


	let check_rule_return env =
		(if(false = env.return_seen && env.return_type = Datatype(Bool)) then
			raise(Error("Missing return statement in rule")));
		 true
	let empty_table_initialization = {parent = None; variables = [];}
	let empty_rule_table_initialization = { rules = []}


	let empty_environment = {return_type = Datatype(String); return_seen = false;
			location = "Setup"; global_scope = empty_table_initialization;
local_scope =
			empty_table_initialization; rule_scope =
empty_rule_table_initialization}


	let find_global_variable env name =
		try List.find(fun (s,_,_) -> s=name) env.global_scope.variables with
		Not_found -> raise Not_found
	let rec check_stmt env stmt = match stmt with
		Block(stmt_list) -> let new_env = env in let getter(env, acc) s = let
				(st, ne) = check_stmt env s in (ne, st::acc) in let
(ls, st) =
					List.fold_left(fun e s -> getter e s)
(new_env, []) stmt_list in
```

```
                                        let revst = List.rev st in
(SBlock(revst), ls)
              | Expr (e) -> let _  = check_expr env e in
                                    (SExpr(get_sexpr env e), env)
              | Return(e) -> let t1 = check_expr env e in (if not((t1 =
                                     env.return_type)) then
raise(Error("Incompatible Return
                                    Type")));
                                    let new_env = {env with return_seen = true} in
                                    (SReturn(get_sexpr env e), new_env)
             | If(e,s1,s2) -> let t = get_type(check_expr env e) in (if
                                    not(t = Bool) then raise(Error("If Statement
must be of type
                                    Bool")));
                                    let (st1, new_env1) = check_stmt env s1
                                    and (st2, new_env2) = check_stmt env s2 in
                                    let ret_seen = (new_env1.return_seen &&
                                    new_env2.return_seen) in
                                    let new_env = {env with return_seen =
ret_seen}
                                    in (SIf((get_sexpr env e), st1, st2), new_env)
            | Loop(e1, s1) -> let t=get_type(check_expr env e1) in (if
                    not(t = Bool) then raise(Error("Improper loop statement")));
                    let (st, new_env) = check_stmt env s1 in (SLoop((get_sexpr env
e1),
                    st), new_env)
            | Decl(datatype, e) ->  let decl = Decl(datatype, e) in let (name, ty )
= get_name_type_from_decl
                                     (Decl(datatype,e)) in
                                    let ((_,dt,_), found) = try(fun f -> ((f env
```

```
                                        name), true)) find_global_variable with
Not_found
                                        ->  ((name, Datatype(ty), None) , false) in
                                        let ret = if(found = false) then
                                                match e with
                                                Id(s) -> let sdecl = get_sdecl env
decl
                                                        in let (n, t, v) =
                                                        get_name_type_value_from_decl
decl
                                                        in let new_env =
add_to_global_table env
                                                        n (Datatype(t)) None in
(sdecl,
                                                        new_env)
                                                | Assign(s1, e1)  ->
                                                let t1 =
                                                        datatype
                                                and t2 =
                                                        get_type(check_expr
                                                        env e1)  in


                                                if(t1 = t2) then let sdecl =
                                                        get_sdecl env decl
                                                in let (n,t,v) =
                                                        get_name_type_value_from_decl
                                                        decl in
                                                let new_env = add_to_global_table env
n
                                                (Datatype(t)) None in
```

```
                                        (sdecl, new_env)
                                        else raise(Error("Type mismatch"))
                                        | _-> raise(Error("Bad variable
                                        declaration"))
                                  else
                                        raise(Error("Already Declared this
                                  variable")) in ret
            | NextPlayer -> (SNextPlayer, env)
      let get_sstmt_list env stmt_list = List.fold_left (fun (sstmt_list, env) stmt
->
            let(sstmt, new_env) = check_stmt env stmt in (sstmt::sstmt_list,
            new_env)) ([],env) stmt_list


      let add_rule env srule_decl =
            let r_table = env.rule_scope in
            let old_rules = r_table.rules in match
                  srule_decl with
                        SRules_Decl(srules_t, datatype) -> let rule_name =
                              srules_t.srname  in let
                              rule_body = srules_t.srbody in let new_rules =
                                    (rule_name, rule_body) :: old_rules in
                              let new_rule_scope = {rules = new_rules} in
                              let final_env = {env with rule_scope =
                                    new_rule_scope} in final_env


      (*semantic checking on rule *)
      let check_rule env rule_declaration =
```

```
              let new_local_scope = {parent = Some(env.local_scope); variables =
[];}

              in

              let new_env = {return_type = Datatype(Bool); return_seen = false;

              location = "in_rule"; global_scope = env.global_scope; local_scope =

                      new_local_scope; rule_scope = env.rule_scope} in

              let (typed_statements, final_env) = get_sstmt_list new_env

              rule_declaration.rbody in

              let _ = check_rule_return final_env  in

              let sruledecl = ({ srname = rule_declaration.rname; srbody =

                      typed_statements}) in

              (SRules_Decl(sruledecl, Datatype(Bool)), env)

      let initialize_rules env rule_list = let (typed_rules, last_env) =

              List.fold_left (fun( sruledecl_list, env) rule -> let (sruledecl, _) =

                      check_rule env rule in let final_env = add_rule env sruledecl
in

                      (sruledecl::sruledecl_list, final_env))([], env) rule_list in

      (typed_rules, last_env)




      let check_ssetup_decl env e = match e with

              Setbd (m) -> let sm = {srows = m.rows; scols = m.cols;} in

                              (SSetbd(sm), env)

              |Setpc(p) -> let sp = {sowner = p.owner; sname = p.name; snum = p.num;

                               sptval = p.ptval;scloc ={ sxc = get_sexpr env
p.cloc.xc; syc =

                                       get_sexpr env p.cloc.yc}} in (SSetpc(sp), env)

              |Setplr(p) -> let sp = {splrname = p.plrname} in (SSetplr(sp), env)

              | Stmt(s) -> let (typed_stmt, final_env) = check_stmt env s in
```

```
                                (SStmt(typed_stmt), final_env)


        (* goes to check setup section *)

       let get_ssetup_decl_list env sdecl_list =

               List.fold_left ( fun (ssetup_decl_list, env) stmt -> let(sstmt,
new_env)

               = check_ssetup_decl env stmt in (sstmt::ssetup_decl_list,
new_env))([], env)

               sdecl_list


       let get_rules_names env rules_list =

               let new_env2 =

               List.fold_left (fun  env rule -> add_to_global_table

               env rule.rname (Datatype(Bool)) None) env rules_list in

               new_env2


       let check_program program =


               let (setup, rules, play) = program in

               let env = empty_environment in

               let (typed_setup, new_env) = get_ssetup_decl_list env setup in (*
should

               check setup section completely *)

               let env2 = empty_environment in

               let env2 = add_to_global_table env2 "Board" (Datatype(Tile)) None in

               let env2 = add_to_global_table env2 "Player" (Datatype(Player)) None
in

               let env2 = add_to_global_table env2 "Pieces" (Datatype(Piece)) None in

               let new_env2 = get_rules_names env2 rules in (* should add rules to
```

```
               global table in the env*)
               let (typed_play, new_env3) =  get_sstmt_list new_env2 play in (*
should
               type check play section *)
               let (typed_rules, new_env4) = initialize_rules new_env3 rules in
               (typed_setup, typed_rules, typed_play)
```

codegen.ml

```
      open Sast
      open Ast
      open Printf
      open Semantics


      let rec jexpr = function
         SLint(l,d) -> string_of_int l
       | SLfloat(f,d) -> string_of_float f
       | SLbool(b,d) -> string_of_bool b
       | SLstring(st,d) -> st
       | SId(s,scope, d) -> (match (String.get s 0) with
            'R' -> s^"()"
            | _ -> s)
       | SBinop(e1, o, e2, d) -> jexpr e1 ^
            (match o with
              Add -> "+ "^jexpr e2 | Sub -> "- "^jexpr e2 | Mult -> "* "^jexpr e2
             | Div -> "/ "^jexpr e2| Equal -> "== "^jexpr e2 | Neq -> "!= "^jexpr e2
```

```
                    | Less -> "< "^jexpr e2 | Leq -> "<= "^jexpr e2 | Greater -> ">
"^jexpr e2
             | Geq -> ">= "^jexpr e2| Or -> "|| "^jexpr e2 | And -> "&& "^ jexpr e2
             | Dequal -> ".equals(" ^ jexpr e2 ^")")
       | SCat(e1,e2,d) -> jexpr e1 ^ "+" ^ jexpr e2


       | SThrough(e1,e2,d) -> "for (int IND=" ^ jexpr e1^ "; IND<" ^
             jexpr e2^"; IND++) {\n"


       | SIncr(e,i,d) ->
             jexpr e ^
             (match i with
               Plus -> "++" | Minus -> "--" )


       | SAssign(v,e,scope,d) ->  v ^ " = " ^ jexpr e


       | SCall(func,args,scope,d) ->
             (match Ast.string_of_expr func with
             "move" -> (match args with
                   [pc; x; y] -> "PC = " ^ jexpr pc^ ";\nPC.loc.x = " ^ jexpr x ^
                         ";\nPC.loc.y = " ^ jexpr y ^ ";\n"
                   | _ -> "Invalid Move Arguments" )
             | "add" -> (match args with
                   [pc; x; y] -> "PC = " ^ jexpr pc^ ";\nPC.loc.x = " ^ jexpr x ^
                         ";\nPC.loc.y = " ^ jexpr y
                   | _ -> "Invalid Add Arguments" )
             | "Input" -> (match args with
                   [a] -> jexpr a ^ " = " ^ (match a with
```

```
                        SId(v,sc,dt) -> (match dt with

                              Datatype(Int) ->
"Integer.parseInt(input.nextLine());"

                              | Datatype(String) -> "input.nextLine();"

                              | Datatype(Float) ->
"Double.parseDouble(input.nextLine());"

                              | _ -> "Cannot Accept Input")

                        | _ -> "Bad Input" )

                  | _ -> "Invalid Input Arguments" )

            | "Output" -> (match args with

                  [a] -> "System.out.println(" ^ jexpr a ^ ")"

                  | _ -> "Invalid Output Arguments")

            | "EndGame" -> (match args with

                  [a] -> "System.out.println(" ^ jexpr a ^ "); System.exit(0)"

                  | _ -> "Invalid EndGame Arguments")

            | _ -> "blah")


      | SBaccess(e1,c,d) -> "PCS.get( Crd(PCS," ^ jexpr c.sxc ^ "," ^

            jexpr c.syc ^ ") )"

      | SAccess(key,pos,d) -> print_string ("SACCESS" ^jexpr key); (match jexpr key
with

            "Player" -> "Players.get(" ^ jexpr pos ^ ")"

            | "Pieces" -> print_string "Pieces!!!!"; "Piece test"

            | "Board" -> print_string "Board!!!!!"; "Board test"

            | _ -> "Invalid Access" )


      | SDaccess(e1,e2,d) ->

            (match e1 with

            SBaccess(expr,coord,d) -> (match jexpr expr with
```

```
                    "Player" -> playerDot e2
                    | "Board" -> boardAccess (jexpr coord.sxc) (jexpr coord.syc) e2
                    | _ -> "Invalid Board Access")
             | SId(keyword,scope,d) -> (match keyword with
                    "Player" -> playerDot e2
                    | _ -> keyword ^
                           (match e2 with
                           SCall(func, args,scope,d) -> (match Ast.string_of_expr
func with
                                  "owner" -> ".owner"
                                  | "name" -> ".name"
                                  | "point" -> ".val"
                                  | "locationx" -> ".loc.x"
                                  | "locationy" -> ".loc.y"
                                  | _ -> "Invalid Pieces field")
                           | _ -> "Invalid Pieces Access") )
             | SAccess(keyword,pos,d) -> (match jexpr keyword with
                    "Player" -> playerAccessDot pos e2
                    | _ -> "")
             | SDaccess(ex1,ex2,d) ->
                    (match ex1 with
                    SBaccess(expr,coord,d) ->
                           (match ex2 with
                           SAccess(keyword,pos,d) ->
                              boardAccessDot (jexpr coord.sxc) (jexpr coord.syc)
keyword pos e2
                           | SCall(func, args,scope,d) ->
                              boardFunctionDot (jexpr coord.sxc) (jexpr coord.syc)
func args e2
```

```
                        | _ -> "Invalid Board Access")
                  | SId(plr,scope,d) -> playerDotInvFunc [] ^
                        (match e2 with
                        SCall(func, args,scope,d) -> (match Ast.string_of_expr
func with
                              "owner" -> ".owner"
                              | "name" -> ".name"
                              | "point" -> ".val"
                              | "locationx" -> ".loc.x"
                              | "locationy" -> ".loc.y"
                              | _ -> "Invalid Pieces field")
                        | _ -> "Invalid Pieces Access")

                  | _ -> "Invalid Board Dot Argument" )

            | SCall(func,args,scope,d) -> (match Ast.string_of_expr func with
                  "CurrentPlayer" -> "Players.get(curPlayer)"
                  | _ -> "")
            | _ -> "Invalid Left Dot Access2" )

       | SNoexpr -> ""

      and playerAccessDot plr_pos e2 =
            (match e2 with
            SCall(func,args,scope,d) -> (match Ast.string_of_expr func with
                  "name" -> "Player.get(" ^ jexpr plr_pos ^ ")"
                  | "inventory" -> "PCS.get( Crd_Plr_Pos(PCS,0,0,Players.get(" ^
                        jexpr plr_pos ^ "-1), 1 ) )"
```

```
                    | "onBoard" -> "PCS.get( Crd_Plr_Pos_Gt(PCS,0,0,Players.get(" ^
                          jexpr plr_pos ^ "-1), 1) )"
                    | _ -> "Invalid Player Function Access" )
             | SAccess(keyword, pos,d) -> (match jexpr keyword with
                    "inventory" -> "PCS.get( Crd_Plr_Pos(PCS,0,0,Players.get(" ^
                          jexpr plr_pos ^ "-1)," ^ jexpr pos ^ ") )"
                    | "onBoard" -> "PCS.get( Crd_Plr_Pos_Gt(PCS,0,0,Players.get(" ^
                          jexpr plr_pos ^ "-1)," ^ jexpr pos ^ ") )"
                    | _ -> "Invalid Player Array Access" )
             | _ -> "Invalid Player Access" )


      and playerDot e2 =
             (match e2 with
             SCall(func,args,scope,d) -> (match Ast.string_of_expr func with
                    "name" -> "Players.get(curPlayer)"
                    | "inventory" -> playerDotInvFunc args
                    | "onBoard" -> playerDotBdFunc args
                    | _ -> "Invalid Player Function Access" )
             | SAccess(keyword, pos,d) -> (match jexpr keyword with
                    "inventory" -> "PCS.get(
Crd_Plr_Pos(PCS,0,0,Players.get(curPlayer)," ^
                          jexpr pos ^ ") )"
                    | "onBoard" -> "PCS.get(
Crd_Plr_Pos_Gt(PCS,0,0,Players.get(curPlayer)," ^
                          jexpr pos ^ ") )"
                    | _ -> "Invalid Player Array Access" )
             | _ -> "Invalid Player Access" )
```

```
      and playerDotInvFunc args  =
            (match args with
            [] -> "PCS.get( Crd_Plr_Pos(PCS,0,0,Players.get(curPlayer),1) )"
            | [pc_n] -> "PCS.get( Crd_Plr_Pcn(PCS,0,0,Players.get(curPlayer)," ^
                  jexpr pc_n ^") )"
            | [pl_n; pc_n] -> "PCS.get( Crd_Plr_Pcn(PCS,0,0,Players.get(curPlayer),"
^
                  jexpr pc_n ^") )"
            | [pl_n; pc_n; c] -> "PCS.get(
Crd_Plr_Pcn(PCS,0,0,Players.get(curPlayer)," ^
                  jexpr pc_n ^") )"
            | _ -> "Invalid Player Access Function")



      and playerDotBdFunc args =
            (match args with
            [pc_n] -> "PCS.get( Crd_Pcn_Gt(PCS,0,0," ^ jexpr pc_n ^") )"
            | [pl_n; pc_n] -> "PCS.get( Crd_Plr_Pcn_Gt(PCS,0,0," ^ jexpr pl_n ^ ","
^ jexpr pc_n ^") )"
            | [pl_n; pc_n; x;y] -> "PCS.get( Crd_Plr_Pcn(PCS," ^ jexpr x ^ "," ^
                  jexpr y ^ "," ^ jexpr pl_n ^ "," ^ jexpr pc_n ^") )"
            | _ ->  "Invalid Player Access Function")


      and boardAccess x y right =
            (match right with
            SAccess(keyword,pos,d) -> boardAccessLoc x y keyword pos
            | SCall(func, args,scope,d) -> boardFunction x y func args
```

```
              | SDaccess(lftexpr, rtexpr,d) ->
                    (match lftexpr with
                    SAccess(keyword,pos,d) -> boardAccessDot x y keyword pos rtexpr
                    | SCall(func, args,scope,d) -> boardFunctionDot x y func args
rtexpr
                    | _ -> "Invalid Board Access")
              | _ -> "Invalid Board Argument" )



       and boardFunctionDot x y func args rtexpr =
              (match Ast.string_of_expr func with
              "Pieces" -> (boardFunctionPieces x y args) ^
                    (match rtexpr with
                    SCall(func, args,scope,d) -> (match Ast.string_of_expr func with
                          "owner" -> ".owner"
                          | "name" -> ".name"
                          | "point" -> ".val"
                          | "locationx" -> ".loc.x"
                          | "locationy" -> ".loc.y"
                          | _ -> "Invalid Pieces field")
                    | _ -> "Invalid Pieces Access")
              | _ -> "Invalid Board Access Function")


       and boardAccessDot x y key pos rtexpr =
              (match jexpr key with
              "Pieces" -> boardAccessLoc x y key pos ^
                    (match rtexpr with
                    SCall(func, args,scope,d) -> (match Ast.string_of_expr func with
```

```
                        "owner" -> ".owner"
                        | "name" -> ".name"
                        | "point" -> ".val"
                        | "locationx" -> ".loc.x"
                        | "locationy" -> ".loc.y"
                        | _ -> "Invalid Pieces field")
                  | _ -> "Invalid Pieces Access")
            | _ -> "Invalid Board Access Function")



      and boardFunctionPieces x y args =
            (match args with
            [pc_n] -> "PCS.get( Crd_Pcn(PCS," ^ x ^ "," ^ y ^ "," ^ jexpr pc_n ^")
)"
            | [pl_n; pc_n] -> "PCS.get( Crd_Plr_Pcn(PCS," ^ x ^ "," ^ y ^ "," ^
jexpr pl_n ^
                  "," ^ jexpr pc_n ^") )"
            | [pl_n; pc_n; c] -> "PCS.get( Crd_Plr_Pcn(PCS," ^ x ^ "," ^ y ^ "," ^
                  jexpr pl_n ^ "," ^ jexpr pc_n ^") )"
            | _ ->  "Invalid Board Function" )



      and boardFunction x y func args =
            (match Ast.string_of_expr func with
            "unoccupied" -> "(Crd(PCS,"^x^","^y^")==-1)"
            | "Pieces" -> boardFunctionPieces x y args
            | _ -> "Invalid Board Function" )
```

```
      and boardAccessLoc x y keyword pos =
            (match jexpr keyword with
            "Pieces" -> "PCS.get( Crd_Pos(PCS," ^ x ^ "," ^ y ^ "," ^  jexpr pos ^
") )"
            | _ -> "Cannot Access Part of Board" )



      let rec jstmt = function
         SBlock(stmts) ->
            "{\n" ^ String.concat "" (List.map jstmt stmts) ^ "}\n"

       | SExpr(expr) -> jexpr expr ^ ";\n";

       | SReturn(expr) -> "return " ^ jexpr expr ^ ";\n";

       | SIf(e, s, SBlock([])) -> "if (" ^ jexpr e ^ ")\n" ^ jstmt s

       | SIf(e, s1, s2) ->  "if (" ^ jexpr e ^ ")\n" ^
            jstmt s1 ^ "else\n" ^ jstmt s2

       | SLoop(e, s) ->
            (match e with
               SThrough(e1,e2,d) -> "for(int IND=" ^ jexpr e1 ^ "; IND<=" ^
                           jexpr e2 ^ "; IND++)" ^ jstmt s
            | e -> "while (" ^ jexpr e ^ ") " ^ jstmt s)

       | SDecl(bgtype,expr,scope) ->
            (match scope with
```

```
              Global -> ""
              | _ -> declare bgtype expr )


       | SNextPlayer -> "NP();"


      and jrulestmt stmt = (match stmt with
              SDecl(bgtype,expr,scope) -> declare bgtype expr
              | _ -> jstmt stmt )




      and declare bgtype expr =
              (match bgtype with
                Datatype(Int) -> "int" | Datatype(Float) -> "double" | Datatype(Bool)
-> "boolean"
              | Datatype(String) -> "String"
              | Datatype(Piece) -> "Pieces"
              | Datatype(Player) -> ""
              | Datatype(Tile) -> ""
              | Datatype(Rule) -> "" ) ^ " " ^
              (match expr with
              SAssign(v,ex,sc,d) -> v ^ (match bgtype with
                     Datatype(Piece) -> "= new Pieces(" ^ jexpr ex ^ ");"
                     | Datatype(String) -> "= new String(" ^ jexpr ex ^ ");"
                     | _ -> "= " ^ jexpr ex ^ ";")
              | _ -> jexpr expr ^ ";")
```

```
      and jsetup = function

         SSetbd(m) -> "rows = " ^ string_of_int m.srows ^ "; cols = "

            ^ string_of_int m.scols ^ ";"

        | SSetpc(pc) -> "for(int IND=0; IND<" ^ string_of_int pc.snum^ "; IND++)
{\n" ^

            "Pieces P = new Pieces(" ^ pc.sowner ^ "," ^ pc.sname ^ "," ^

            string_of_int pc.sptval ^ "," ^ jexpr pc.scloc.sxc ^ "," ^

            jexpr pc.scloc.syc ^ ");\nPCS.add(P);}"

       | SSetplr(plr) -> "Players.add(" ^ plr.splrname ^ ");"

       | SStmt(s) -> jstmt s




      and jrules r = (match r with

            SRules_Decl(rule,d) ->

                  "static boolean " ^ rule.srname ^ "() {\n" ^

                  String.concat "\n" (List.rev(List.map jrulestmt rule.srbody)) ^
"}" )


      and jprogSetup setup_list =

            String.concat "\n" (List.rev(List.map jsetup setup_list))


      and jprogRules rule_list =

            String.concat "\n" (List.rev(List.map jrules rule_list))


      and findGlobals play = (match play with

            SDecl(bgtype,expr,scope) ->

                  (match scope with

                  Global -> "public static " ^ declare bgtype expr
```

```
                | _ -> "" )
        | _ -> "")


and jprogram program =
        let (ssetup, srules, sstmt) = program in
        let setup_func = jprogSetup ssetup
        and globals = String.concat "\n" (List.rev(List.map findGlobals sstmt))
        and body = String.concat "" (List.rev(List.map jstmt sstmt))
        and rule_func = jprogRules srules
        in
        sprintf
"
import java.awt.Point;
import java.util.LinkedList;
import java.util.Scanner;

public class BG {

static Scanner input = new Scanner (System.in);
public static LinkedList<String> Players = new LinkedList<String>();
public static LinkedList<Pieces> PCS = new LinkedList<Pieces>();
public static int curPlayer = 0;
public static int rows;
public static int cols;
public static Pieces PC;

static void setup() {
%s
```

```
	}


	%s


	public static void main(String[] args) {
	setup();
	while(true) {
	%s
	}


	}


	%s


	/*BoredGame helper functions*/
	static void NP () {
		curPlayer++;
		if (curPlayer > Players.size()-1) {
			curPlayer=0;
		}
	}
	static int Crd_Plr_SearchCt (LinkedList<Pieces> plist, int x, int y, String
plr) {
		int count = 0;
		Point test = new Point(x,y);
		for (int i=0; i<plist.size(); i++){
			Pieces t = plist.get(i);
			if (t.loc.equals(test)) {
```

```
                        count++;
                }
            }
            return count;
      }
      static int Crd_Plr_SearchCt_Gt (LinkedList<Pieces> plist, int x, int y, String
plr) {
            int count = 0;
            Point test = new Point(x,y);
            for (int i=0; i<plist.size(); i++){
                  Pieces t = plist.get(i);
                  if (t.loc.x > test.x && t.loc.y>test.y) {
                        count++;
                  }
            }
            return count;
      }
      static int Pc_Pos (LinkedList<Pieces> plist, Pieces p, int pos) {
            int count = pos;
            for (int i=0; i<plist.size(); i++){
                  Pieces t = plist.get(i);
                  if (t.equals(p)) {
                        if (count==1) {
                              return i;
                        }
                        count--;
                  }
```

```
            }
            return -1;
      }
      static int Crd_Plr_Pos (LinkedList<Pieces> plist, int x, int y, String plr,
int pos) {
            int count = pos;
            Point test = new Point(x,y);
            for (int i=0; i<plist.size(); i++){
                  Pieces t = plist.get(i);
                  if (t.loc.equals(test) && t.owner.equals(plr)) {
                        if (count==1) {
                              return i;
                        }
                        count--;
                  }
            }
            return -1;
      }
      static int Crd_Plr_Pos_Gt (LinkedList<Pieces> plist, int x, int y, String plr,
int pos) {
            int count = pos;
            Point test = new Point(x,y);
            for (int i=0; i<plist.size(); i++){
                  Pieces t = plist.get(i);
                  if (t.loc.x > test.x && t.loc.y > test.y && t.owner.equals(plr))
{
                        if (count==1) {
                              return i;
                        }
```

```
                        count--;
                }
        }
        return -1;
    }
    static int Crd_Pos (LinkedList<Pieces> plist, int x, int y, int pos) {
        int count = pos;
        Point test = new Point(x,y);
        for (int i=0; i<plist.size(); i++){
                Pieces t = plist.get(i);
                if (t.loc.equals(test)) {
                        if (count==1) {
                                return i;
                        }
                        count--;
                }
        }
        return -1;
    }
    static int Crd_Plr (LinkedList<Pieces> plist, int x, int y, String plr) {
        Point test = new Point(x,y);
        for (int i=0; i<plist.size(); i++){
                Pieces t = plist.get(i);
                if (t.loc.equals(test) && t.owner.equals(plr)) {
                        return i;
                }
        }
        return -1;
```

```
	}

	static int Crd_Plr_Pcn (LinkedList<Pieces> plist, int x, int y, String plr,
String pcn) {

		Point test = new Point(x,y);

		for (int i=0; i<plist.size(); i++){

			Pieces t = plist.get(i);

			if (t.loc.equals(test) && t.owner.equals(plr) &&
t.name.equals(pcn)) {

				return i;

			}

		}

		return -1;

	}

	static int Pcn (LinkedList<Pieces> plist, String pcn) {

		for (int i=0; i<plist.size(); i++){

			Pieces t = plist.get(i);

			if (t.name.equals(pcn)) {

				return i;

			}

		}

		return -1;

	}

	static int Crd_Pcn_Gt (LinkedList<Pieces> plist, int x, int y, String pn) {

		Point test = new Point(x,y);

		for (int i=0; i<plist.size(); i++){

			Pieces t = plist.get(i);

			if (t.loc.x > test.x && t.loc.y > test.y && t.name.equals(pn)) {

				return i;
```

```
                  }

            }

            return -1;

      }

      static int Crd_Plr_Pcn_Gt (LinkedList<Pieces> plist, int x, int y,String
plr,String pn) {

            Point test = new Point(x,y);

            for (int i=0; i<plist.size(); i++){

                  Pieces t = plist.get(i);

                  if (t.loc.x>test.x && t.loc.y>test.y && t.owner.equals(plr) &&
t.name.equals(pn)) {

                        return i;

                  }



            }

            return -1;

      }

      static int Plr_Pcn (LinkedList<Pieces> plist, String plr, String pcn) {

            for (int i=0; i<plist.size(); i++){

                  Pieces t = plist.get(i);

                  if (t.name.equals(pcn) && t.owner.equals(plr)) {

                        return i;

                  }

            }

            return -1;

      }

      static int Crd_Pcn (LinkedList<Pieces> plist, int x, int y, String pcn) {

            Point test = new Point(x,y);
```

```
		for (int i=0; i<plist.size(); i++){
			Pieces t = plist.get(i);
			if (t.loc.equals(test) && t.name.equals(pcn)) {
				return i;
			}
		}
		return -1;
	}
	static int Crd (LinkedList<Pieces> plist, int x, int y) {
		Point test = new Point(x,y);
		for (int i=0; i<plist.size(); i++){
			Pieces t = plist.get(i);
			if (t.loc.equals(test)) {
				return i;
			}
		}
		return -1;
	}
	static int Crd_Plr_Gt (LinkedList<Pieces> plist, int x, int y,String plr) {
		Point test = new Point(x,y);
		for (int i=0; i<plist.size(); i++){
			Pieces t = plist.get(i);
			if (t.loc.x>test.x && t.loc.y>test.y && t.owner.equals(plr)) {
				return i;
			}

		}
		return -1;
```

```
	}


	}
	" setup_func globals body rule_func
```

compiler.ml

```
	let _ =
	let lexbuf = Lexing.from_channel stdin in
	let ast = Parser.program Scanner.token lexbuf in
	(*let listing = Ast.string_of_program ast
	in print_string listing in *)
	let sast = Semantics.check_program ast in
	let pgm = Codegen.jprogram sast in
	let output = open_out "BG.java" in
	output_string output pgm
	(*print_string "done \n"*)
```

Pieces.java

```
	import java.awt.Point;
	public class Pieces {
		public String owner;
		public String name;
		public int val;
		public Point loc;
```

```
	Pieces(String own, String nm, int v, int x, int y) {
		owner = new String(own);
		name = new String(nm);
		val = v;
		loc = new Point(x,y);
	}

	public boolean equals(Pieces p) {
		boolean o = this.owner.equals(p.owner);
		boolean n = this.name.equals(p.name);
		boolean v =  this.val == p.val;
		boolean l =  this.loc.equals(p.loc);
		return (o&&n&&v&&l);
	}

	public int hashCode() {
		return 0;
	}
}
```

Makefile

```
.PHONY : compile
compile:
	ocamlc -c ast.ml
	ocamlyacc parser.mly
```

```
		ocamlc -c parser.mli

		ocamlc -c parser.ml

		ocamllex scanner.mll

		ocamlc -c scanner.ml

		ocamlc -c sast.ml

		ocamlc -c semantics.ml

		ocamlc -c codegen.ml

		ocamlc -c compiler.ml

		ocamlc -o compiler ast.cmo parser.cmo scanner.cmo sast.cmo semantics.cmo
codegen.cmo compiler.cmo

	clean:

		rm -f parser.ml ast.cmi ast.cmo compiler.cmi compiler.cmo parser.cmi
parser.cmo parser.mli parser.ml sast.cmi sast.cmo scanner.cmi scanner.cmo scanner.ml
semantics.cmi semantics.cmo codegen.cmo codegen.cmi compiler BG.java BG.class
```

run.sh

```
	./compiler < $1

	javac BG.java

	java BG
```

tests.sh

```
	./run.sh tests/rules.board

	./run.sh tests/greaterthantest.board

	./run.sh tests/rules.board

	./run.sh tests/elseiftest.board

	./run.sh tests/ifelsetest.board
```

```
./run.sh tests/piecetest.board

./run.sh tests/playertest.board

./run.sh tests/binop.board

./run.sh tests/ruleCheck.board

./run.sh tests/helloworld.board

./run.sh tests/wrongtypecast.board

./run.sh tests/tictactoe.board
```

helloworld.board

```
Setup{

 new Board(3,3);


}

Rules{


}

Play{

EndGame("Hello World");

}
```

tictactoe.board

```
Setup{

       new Board(3,3);

       new Player("A");

       new Player("B");
```

```
       new Pieces("A", "X", 6);
       new Pieces("B", "O", 6);

}
Rules{

 rule R1: if(x < 4 && x > 0) { return true; } else { return false;};
rule R2: if(y < 4 && y > 0) { return true; } else { return false;};

 rule R3: if(R1 && R2 && R4) { return true;} else {return false;};
 rule R4: return Board[(x,y)].unoccupied();;
rule R5: int i = 1;
       int j = 1;
        bool full = true;
       loop(i < 4) {
       j=1;
       loop (j < 4) {
       if (Board[(i,j)].unoccupied()) {
       full = false; }
       j = j+1; }
       i = i + 1; }
        return full;;
rule R6: int k = 1;
                int h = 1;
                int val = 0;
                bool win = false;
                /* every row */
                loop(k<4) {
```

```
                        /* every col */
                        h=1;
                        loop(h<4) {
                              if (Board[(h,k)].unoccupied()) {
                                    win = false;
                              }
                              else {
                              if (Board[(h,k)].Pieces[1].owner() .= Player.name()
) {/* checks if    the owner of the first piece on the board at i,j is owned by the
current Player*/
                                    val = val+1;
                              }
                              else {
                                    val = val;
                              }}
                              h = h+1;
                        }
                        if (val==3) {
                              win = true;
                              return win;
                        }
                        else {
                              val = 0;
                              k = k+1;
                        }
                     }
                     return win;;
      rule R7: int m7 = 1;
```

```
                        int l7 = 1;
                        int val7 = 0;
                        bool win7 = false;
                        /* every row */
                        loop(m7<4) {
                            /* every col */
                            l7=1;
                            loop(l7<4) {
                                  if (Board[(m7,l7)].unoccupied()) {
                                        win7 = false;
                                  }
                                  else {
                                  if (Board[(m7,l7)].Pieces[1].owner() .=
Player.name() ) {/* checks if    the owner of the first piece on the board at i,j is
owned by the current Player*/
                                        val7 = val7+1;
                                  }
                                  else {
                                        val7 = val7;
                                  }}
                                  l7 = l7+1;
                            }
                            if (val7==3) {
                                  win7 = true;
                                  return win7;
                            }
                            else {
                                  val7 = 0;
```

```
                                m7 = m7+1;

                            }

                        }

                        return win7;;


      /* diagonal win */

             rule R8: int n=1;

                      int a=0;

                      loop(1:3)/*goes through each row*/

                      {

                           if (Board[(n,n)].unoccupied() == false &&
Board[(n,n)].Pieces[1].name() .= Player.inventory().name()){ /*checks if the first
piece on the board at location i,j has the same name as the first piece in the
current player's inventory*/

                                a = a+1;

                           }

                           n = n + 1;

                      }

                      if (a==3){

                           return true;

                      }

                      else{

                           n = 1;

                           a = 0;

                           loop(1:3){

                           if (Board[(n,4 - n)].unoccupied() == false && Board[(n, 4
- n)].Pieces[1].name() .= Player.inventory().name()){

                                a = a +1;

                           }
```

```
                        n = n + 1;
                        }
                        if( a==3){
                               return true;
                        }
                        return false;
                    };
    /*checks if any of the three win contions are satisfied*/
           rule R9: if(R6 || R7 || R8) {
                        return true;
                    }
                    else {
                        return false;
                    };



    }
    Play{
           string output;
           int q = 1;
           q =1;
    loop(1:3){/*displays the current board on the screen*/
           if(Board[(1,q)].unoccupied()){
                  output =  " _ ";
           }
           else{
                  output = Board[(1,q)].Pieces[1].name();
           }
```

```
            if(Board[(2,q)].unoccupied()){
                  output = output ^  " _ ";
            }
            else{
                  output = output ^ Board[(2,q)].Pieces[1].name();
            }
            if(Board[(3,q)].unoccupied()){
                  output = output ^  " _ ";
            }
            else{
                  output = output ^ Board[(3,q)].Pieces[1].name();
            }
            Output(output);
              q++;
      }



            int x;
            int y;

            Input(x);/*takes keyboard input from the user and stores it in the
coordinate in*/
            Input(y);
            if(R3){
                  add(Player.inventory(), x,y);
                  if(R9){
                        EndGame("Player " ^ Player.name() ^ " Won");  /* break,
display board and output  message */
```

```
                    }
                    else{
                           if(R6){
                                  EndGame("Board Full! No Winner!"); /* break, display
board and output message*/
                           }
                    }
                    NextPlayer;
             }
```