GOBLIN

Final Report

Kevin Xiao, kkx2101

Bayard Neville, ban2117

Christina Floristean, cf2469

Gabriel Uribe, gu2124

Table of Contents

1. Introduction

Goblin is a language designed to allow anyone to make their own turn-based game without extensive knowledge of software development. It will follow a simplified object oriented model that hews as closely as possible to the familiar way things act in real life. All programs create a playable turn-based game where a user is placed on a rectangular world created by ASCII characters. There is a feed of recent actions and their results as well as a section for displaying stats like health or money. All games compile to LLVM and follow the same basic loop structure: entities on the map move, then print the map, action feed, and stats.

2. Language Tutorial

Compilation Instructions

We created a compilation script called *gobble*, which takes a goblin source file as input and generates an executable game. The gobble script runs the source file through our translator (goblin.native) to generate LLVM IR. LLVM IR for the necessary C dependencies is also generated. Once the goblin program and C dependencies have been converted to LLVM, clang is used to link the files and generate an executable.

To compile a program using the provided goblin script, use the following commands:

```
./gobble game.gob
```

To compile a program without using the script:

```
cat game.gob | ./goblin.native > game.ll
clang -emit-llvm -S -c csrc/goblin_helper.c
clang  game.ll goblin_helper.ll -o goblinGame
```

This will generate an executable called goblinGame

Creating a Basic Goblin Program

Every Goblin program is divided into three sections: world, entities, and functions. The structure of a program that moves a player based on w,a,s,d,q user input is as follows:

```
/* Create and initialize world. */
world[20,40] {
    place(player, 10, 30);
```

```
}

/* Define entities. */
entities {

  /* Associate a symbol with each entity (@ for player). */
  @:player {

    /* Declare player fields. */
    num health;

    /* Constructor for player; initialize fields. */
    build {
      this.health = 5;
    }

    /* Instructions to execute every round. */
    does {
      playerMove(this);
      prints("Press q to quit or any other key to end turn");
      if(getKey() == 'q') {
        exit;
      }
    }
  }
}

/* Place all functions in 'functions' block. */
functions {

  /* Function to take in user input and move player around the board. */
  bool playerMove(entity p) {

    /* Declare all variables. */
    num r;
    num c;
    char k;
    bool moved;

    /* Initialize variables. */
```

```
    moved = false;
    r = row(p); /* Built in function row(). */
    c = col(p); /* Built in function col(). */
    prints("Press wasd to move or q to quit");

    /* Wait for user input. */
    while(not moved) {
      k = getKey(); /* Built in function getKey(). */
      if(k == 'w' and r > 0) {
        move(p, r - 1, c); /* Built in function move(). */
        moved = true;
      } else if(k == 'a' and c > 0) {
        move(p, r, c - 1);
        moved = true;
      } else if(k == 's' and r < rows-1) {
        move(p, r + 1, c);
        moved = true;
      } else if(k == 'd' and c < cols-1) {
        move(p, r, c + 1);
        moved = true;

        /* Exit game loop and end game. */
      } else if(k == 'q') {
        exit;
        moved = true;
      }
    }
    return true;
  }

}
```

Looking at this example, we see that world must be given a defined height and width. The block itself is a constructor for the initial board setup. In this example, we place player at a specific location on the board. We only have one entity called player in this example which will appear on the board as symbol '@'. Within player, field variables must be declared before the optional build and does blocks. The fields can be initialized in the build block, which acts as a constructor for the player. Build is only called once per entity instantiation. The does block, however, is called every iteration of the game loop. Keyword 'this' is used to refer to the instance of player.

Function playerMove(this) is called every turn. This function can be found in the functions block. It awaits user input to move the player one step up, down, left, or right on the board, using the built-in function getKey().

3. Language Manual

Lexical Conventions

Identifiers

Identifiers are strings used for naming variables, functions, entities, and worlds. Identifiers are case sensitive, can contain letters, numbers, and underscores, but must start with a letter.

Keywords

Keywords are reserved by the language and cannot be used as identifiers. Keywords are case sensitive:

and	or	not	true	false	is
for	if	else	while	return	exit
functions	entities	world	build	does	num
char	bool	null	entity	this	empty
rows	cols				

Literals

String Literals

A string literal is a sequence of zero or more ASCII characters. String literals are enclosed in single or double quotes.

Number Literals

A number literal is an optional minus sign followed by a sequence of decimal digits.

Delimiters

Spaces, tabs, and new-line characters separate tokens and can be used to format source code but have no functional value. Whitespace is completely ignored by the compiler.

Comments

Comments are enclosed by /* and */. Comments can span multiple lines. Comments are ignored by the compiler.

Parentheses and Braces

Parentheses are used to enclose the arguments of a function or to alter precedence of expression evaluation (See Expressions and Operators below). Braces are used to compartmentalize code blocks under definitions like entities, functions, worlds, etc.

Commas

Commas separate function arguments.

Semicolons

Semicolons punctuate the end of expressions, declarations, initializations, and assignments.

Types

Primitive Data Types	**Definition**
num	A 32 bit signed integer with a range from -2,147,483,648 to 2,147,483,647. The keyword is num.
bool	A 1 bit true or false value. The keyword is bool.
char	A single ASCII character enclosed within single quotes. This character is represented by a single unsigned 8 bit integer.

Non-Primitive Data Types	**Definition**
Entity	A datatype representing a player, character, or object. An entity must be initialized with an ASCII character to represent it in the world.

Expressions and Operators

Operator Precedence

All Goblin operators have well defined precedence. Operators with equal precedence are evaluated from left to right, otherwise expressions are evaluated from highest to lowest precedence. Operator precedence can be explicitly defined by the programmer using parenthesis. Parenthesis are used to create subexpressions, which have their own scope of precedence and are evaluated before the outer expression. Expressions can be nested using parentheses and are evaluated from the innermost subexpression outwards. If there are no nested expressions, subexpressions are evaluated from left to right.

1. Arithmetic Operators (multiplication, division, modulo, addition, subtraction):

```
*, /, %, +, -
```

Arithmetic operators have the traditional order of precedence and associativity, operate on numbers and return numbers. The multiplication, division, addition and subtraction operators are associative; the modulo operator is non-associative. Multiplication, division, and modulo have equal precedence and a greater precedence than addition and subtraction, which also have equal precedence.

2. Logical Operators:

```
and, or, not
```

Logical operators operate on Boolean values or expressions that evaluate to Boolean values are return a Boolean value. The logical operators `and` and `or` are associative and operate on two boolean expressions:

```
<boolean_expression> and <boolean_expression>

<boolean_expression> or <boolean_expression>
```

`and` returns true if both boolean expressions evaluate to true and false otherwise. `or` returns true if one or both of the boolean expressions evaluate to true and false otherwise. `and` and `or` have equal precedence but a lower precedence than the `not` operator. `not` is a unary operator that negates and returns the value of the given boolean expression, it operates on the boolean expression to the right of the operator.

```
not <boolean_expression>
```

3. Assignment (Assign, Assign and increment, Assign and decrement, Multiply and assign, Divide and assign):

```
=, +=, -=, *=, /=
```

Assignment operators are used to assign a value to a variable, such as:

```
num number;

number = 9;
```

Variables are strongly typed, meaning the type of the variable must be declared when created. However, it is unnecessary and syntactically invalid to specify the type every time the variable is used. It is possible to set a variable to the evaluation of an expression, such as:

```
number = 4 + 3; /*also an example of re-assigning a variable*/
```

Increment and assign and decrement and assign are used in a similar way, either incrementing the number value of variable or decrementing the number value of a variable by the number that is given as a result when the expression to the right of the operator is evaluated.

```
<variable> += <value to increment by>

<variable> -= <value to decrement by>

number += 3; /*number, previously set to 7, is now equal to
10*/
```

Increment and assign and decrement and assign can only be used with variables of type num because it is meaningless to perform an increment or decrement operation on non-numeric types. Furthermore the term to the right of the operator must be of type num because incrementing a number with a non-number gives an undefined result and is therefore not allowed. Parenthesis can be used on the right side of the operator as long as the type of the evaluated expression corresponds to the type declared for the variable. Technically the assignment operators all have equal precedence, however only one assignment operator will be used in a statement to assign a value to a variable so in practice precedence relative to each other is unimportant. The assignment operators have a lower precedence than all other operators.

4. Boolean comparisons (Not equal to, Equal to, Less than, Greater than, Less than or equal to, Greater than or equal to, Type equality):

```
!=, ==, <, >, <=, >=, is
```

Boolean comparison operators are binary operators used to compare two expressions; these operators return a third Boolean value derived from the inputs. The expressions must be comparable types that make sense to use these operators. For example, comparing if a string is

greater than a num is undefined and therefore not allowed. However, comparing a boolean with a boolean or a num with a num is meaningful and therefore allowed.

```
<expression> == <expression>

<expression> != <expression>

<num_expression> >= <num_expression>
```

The `is` operator is the type equality operator. It operates on an entity and an entity type and returns a boolean value.

```
<entity> is <entity_type>
```

The `is` operator returns `true` if the entity on the left of the operator is of the type specified on the right side of the operator and `false` otherwise.

Statements

Basic Statements

The simplest type of statement is an expression followed by a semicolon.

```
<expression>;
prints(“hi”);
a + b;
```

Return Statements

Return statements begin with “return” followed by an optional value, followed by a semicolon. Return statements inside functions return the specified type and exit the function. Return statements in a build or does block must not return a value, but still exit from the build or does block.

```
return <value>; /* In a function with return type matching type of value */
return 5; /* Inside a function with num return type */
```

Exit Statement

The exit statement is used to terminate the program, it has no return value. It can be placed within any code block. Code placed after the exit statement will never be reached because the program terminates immediately upon encountering an `exit` statement.

```
<statements>
exit;
<unreached_statements>
```

For Loops

A for loop consists of a header and a body. The header consists of “for” followed by a parentheses-enclosed, semicolon-separated list of three expressions. The first expression is run once when before the first iteration of the loop and is optional. The second expression is the condition for the loop and is mandatory. The loop keeps running the code in the body until the condition evaluates to false. The third expression is run after each iteration in the loop and is optional. The body is enclosed in curly braces and consists of any number of statements.

```
for(<optional_expression>;<conditionexpression>;<optional_expression>
) {
      <statements>
}
for(i = 0; i < 10; i += 1) { /* i is previously declared */
      e.health -= 1; /* this line is run ten times */
}
```

While Loops

A while loop consists of a header and a body. The header consists of “while” followed by a parentheses-enclosed expression. The expression is the condition for the loop and is mandatory. The loop keeps running the code in the body until the condition evaluates to false. The body is enclosed in curly braces and consists of any number of statements.

```
while(<condition expression>) {
      <statements>
}
while(e.health < 10) {
      e.health += 1;
}
```

If Statements

An if statement consists of a header and a body. The header consists of “if” followed by a parentheses-enclosed expression. The expression is the condition for the if statement. The body is enclosed in curly braces and consists of any number of statements. If the condition evaluates to true then the statements in the body are executed.

```
if(<condition expression>) {
      <statements> /* Executed if the condition evaluates to true */
}
if(e.health > 10) {
      e.health = 10;
```

```
}
```

If Else Statements

If-else statements are the same as if statements, but the body of the if is followed by “else” which is followed by another body enclosed in curly braces. If the condition evaluates to true, then the statements in the first body are executed, like in the if statement, but if the condition evaluates to false, then the statements in the second body are executed. An else is attached to the most recent else-less if statement.

```
if(<condition expression>) {
      <statements> /* Executed if the condition evaluates to true */
} else {
      <statements> /* Executed if the condition evaluates to false */
}
if(e.health > 10) {
      e.health = 10;
} else {
      e.health += 1;
}
```

Else If Statements

By combining if and if-else statements it is possible to create a conditional with more than two possible results. The conditions can cascade into several different possible outcomes.

```
if(e.health > 10) {
      e.health = 10;
} else if(e.health == 5) {
      e.health = 1000;
} else {
      e.health = 100 /* e.health <= 10 and e.health != 5 */
}
```

Program Structure

A Goblin program consists of 3 code blocks, that describe a game that are compiled into a single player game executable. At a very abstracted level a Goblin program looks something like this:

```
/*    Abstract structure of a Goblin program      */

world[<height>, <width>]{
```

```
            <variable declarations>
            <statements>
}
entities{
      <character>:player{
                              <variable declarations>
                              build{
                                    <statements>
                              }
                              does{
                                    <statements>
                              }
      }
      <character>:<entity name>{
                              <variable declarations>
                              build{
                                    <statements>
                              }
                              does{
                                    <statements>
                              }
      }
      ...
}
functions{
      <type> <function name>(<type> <parameter name>, ...) {
            <variable declarations>
            <statements>
            <return statement if return type is not void>
      }
      ...
}
```

The structure of a goblin program is inspired by how we approached the problem of game design. We believe the best way to design a turn based game is in terms of these 3 categories: world, entities, and functions. **A game is always a collection of one or more *entities* that may have *functions* within the game and exist in a *world*.**

All game entities, or objects within the game, are defined within the entities block. Consider the following:

```
entities{
```

```
      <character>:player{
                              <variable declarations>
                              build{
                                    <statements>
                              }
                              does{
                                    <statements>
                              }
      }
      <character>:<entity name>{
                              <variable declarations>
                              build{
                                    <statements>
                              }
                              does{
                                    <statements>
                              }
      }
      ...
}
```

Entities are bound - using the : operator - to a specific character that is used to represent instances of the entity on the game board.

As seen above, it is possible for entities to contain a does block and a build block within their definition. The does block contains the recurring behavior logic of the entity. The build block is a constructor for the object. Code contained within the build block will be executed exactly once at the entity's creation.

Functions within the does block are run every time it is that entity's turn. It is not required for an entity to do anything and there is no limit to how many functions can be listed within a does block. An entity is only allowed to define one does block. All of the functions listed in the build and does blocks are listed under the *functions* code block.

A Goblin game contains one world that specifies its height and width. Global variables 'rows' and 'cols' refer to these values. The contents of the world block function as a constructor for the initial board.

Game Loop

The Goblin compiler generates a turn based game as defined by the source code. However, there are some mechanisms of game operation that are invariant. Every Goblin game consists

of two players, the user and the computer. Each player gets a turn and all turns must happen sequentially. This means that one player cannot take their turn until the previous player has completed their turn. Practically, this means that the game will wait indefinitely for the user to take their turn before allowing the computer to alter the states of any entity or world in the game because the computer will continue playing unless an end state of the game is reached or the player decides to quit. Despite the appearance that the player is given the first turn, the computer is always given the first turn in any game to allow for initial world creation and game setup. After each turn, regardless of if it's a player turn or a computer turn, the screen is redrawn to reflect changes to the game state.

The game loop can be described as follows:

1. Turn taken
2. Screen redraw

The progression of a turn taken by the computer is as follows: the actions for computer controlled objects (meaning on-screen entities) – defined in does blocks - are performed, the states of the computer controlled objects are updated, the screen is redrawn and presented to the user.

The progression of a turn taken by a user is as follows: the user presses a key which getKey() returns, usually in a function called by the player's does block.

The screen is redrawn automatically after every turn. A game loop will continue run unless the programmer reaches an exit condition or quits, which will break the loop and end the game.

Declarations

Variable Declarations

Variables of all types besides entity are declared with a type, followed by the variable name. All variables must be declared at the beginning of the block. Variable assignment occurs after these declarations. The assignment operator is followed by a literal value of the variable's type, followed by a semicolon. Variables of type entity are declared with "entity" followed by the variable name, followed by a semicolon. Using the peek method, the entity is assigned an entity on the board at the specified coordinates.

```
<variable type except for entity> <variable name>;
<variable name> = <value matching the variable type>;
entity <variable name>;
num things;
entity e;
e = peek(3, 3);
things = 10;
```

Function Declarations

Functions consist of a header followed by a body. The header consists of a type, the function name, and a parentheses-enclosed, comma-separated list of any number of parameters. Each parameter is a type and a parameter name. The parameter variables are available inside the function body when the function is called.

The body is enclosed in curly braces and consists of any number of variable declarations followed by any number of statements. There must be a return statement of a type that matches the return type.

```
<type> <function name>(<type> <parameter name>, ...) {

      <variable declarations>
      <statements>
      <return statement>
}
num getDifference(num a, num b) {
      num c;
      c = a - b;
      return c;
}
```

Field Declarations

Field declarations are the same as variable declarations, but they are at the beginning of entity definitions and are owned by that entity. Entity fields can be assigned values within the build block.

Build Block

A build block is essentially a set of initialization statements that belong to an entity. The build block runs once every time an instance of an entity is created. Fields of the entity instance can be referenced by this.<field_name>.

The build block consists of the word “build” followed by the body. The body is enclosed in curly braces and consists of any number of variable initializations followed by any number of statements.

```
<variable declarations>
build {
      <statements>
}
```

```
num bonus;
num health;
build {
      this.bonus = 5;
      this.health += this.bonus;
}
```

Does Block

A does block is essentially a set of function calls that belong to an entity and are executed every iteration of the game loop. Fields of the entity instance can be referenced by this.<field_name>. The does functions for each entity are called from left to right, top to bottom on the board for every turn.

The does block consists of the word “does” followed by the body. The body is enclosed in curly braces and consists of function calls.

```
does {
      <statements>
}

does {
      heal(this);
      move_goblin(this);

}
```

Built-in Functions

Place Function

`bool place(string entity_name, num x, num y)`
Places an instance of an entity of the type that matches the string entity_name in the current world at the location specified by the “x” and “y” values.

Peek Function

`entity peek(num x, num y)`
Returns the entity on the board at the given “x” and “y” values if there is one. Using the “is” operator on the entity returned will determine the type of entity, or if it is empty.

Remove Function

`bool remove(entity e)`

Attempts to remove the entity instance from the board. It returns true if it removes something and false if there is nothing to remove.

Move Function

```
bool move(entity e, num x, num y)
```
Attempts to move the entity instance to another tile on the board specified by x and y. It returns true if successful and false otherwise.

Row Function

```
num row(entity e)
```
Returns the row value for an entity instance.

Col Function

```
num col(entity e)
```
Returns the column value for an entity instance.

Get Key Function

```
char getKey()
```
Return a char of the last key pressed by the user. This is a blocking function, once called it will wait until a key is pressed by the user.

Print Functions

```
bool prints(string text)
```
Prints the given string to the log under the map display.

```
bool printb(bool b)
```
Prints the given boolean value to the log under the map display.

```
bool print(num n)
```
Prints the given integer to the log under the map display.

Scope

Global Scope

A Goblin program contains the global variables rows and cols, which refer to the height and width of the world. Other than these two, no variables have global scope. The different types of entities or functions defined in their respective code blocks are all ‘public’ in that they are visible in every part of the program.

This function called goblinMove shows how the global variables might be used:

```
bool goblinMove(entity p) {
    num r;
    num c;
    entity e;
    r = row(p);
    c = col(p);

    e = peek(r+1, c);
    if (e is empty) {
      move(p, (r+1)%rows, c);
    }

    e = peek(r, c+1);
    if (e is empty) {
      move(p, r, (c+1)%cols);
    }
    return true;
  }
```

Function Scope

Variables defined within a function are only visible and recognized within the function itself. The same logic applies to for loops, while loops, and if/else if/else blocks.

4. Project Plan

Roles & Responsibilities

Kevin was the team manager. He also assisted Bayard, who was the language guru. He had the most github contributions and coded the majority of the codegen and semantic analyzer. Christina was the system architect

Process

For planning and specification, we had meetings twice a week to discuss design and brainstorm. For development and testing, we had meeting three times a week to work together. We all worked together on the scanner, parser, and AST. Primarily, Bayard and Kevin worked on the codegen and the semantic analyzer. Christina completed all of the semantic checking. Gabe completed all of the testing, external C libraries, and linking. Bayard also helped on testing.

Project Timeline

Oct 12 - Nov 6	Scanner, Parser & AST
Nov 6 - 21	Hello World
Nov 21 - Dec 3	Game Loop & Semantic Analyzer
Dec 3 - 6	Arrays
Dec 6 - 10	Primitive World
Dec 9 - 10	Peek & Array Accessor
Dec 10 - 11	Convert Build and Does to Functions
Dec 11 - 14	Place & Array Mutator
Dec 12 - 17	GetKey & User Input
Dec 14 - 17	Structs
Dec 14 - 15	Struct Accessor & Mutator
Dec 14 - 15	Tile Structs
Dec 14 - 15	Row and Column Functions
Dec 15 - 16	Entity Structs
Dec 16 - 17	World Rendering
Dec 16 - 19	Semantic Checking
Dec 17 - 18	Build and Does Invocation
Dec 17 - 18	Move
Dec 17 - 18	Remove & Free
Dec 17 - 18	Exit
Dec 17 - 20	“Goblin Hunter” Sample Program
Dec 18 - 19	Is & Entity Type Checking
Dec 18 - 19	ClearScreen & Terminal Clearing
Dec 18 - 19	Gobble Linking and Compilation Script
Dec 19 - 20	Completed World

Development Environment

We developed Goblin in OCaml and C. C was only used for external libraries for user input and clearing the terminal screen.

Programming Style Guide

Rules:

- 2 spaces indentation
- 140 character line max
- Underscore, not camel case

Project Log

commit 80fe50724c0af10aadfaf1ff2f7737855138ae22
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Tue Dec 20 23:24:24 2016 -0500

```
    bad test fixed

commit dab7b3d522380b1c4a17bef585b05e687ef1125d
Merge: e811eb1 5dfaf62
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Tue Dec 20 23:11:16 2016 -0500

    Merge branch 'master' of https://github.com/bayardneville/goblin

commit e811eb10b0a3d791e2922c5d27a7042a046b0cd6
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Tue Dec 20 23:08:24 2016 -0500

    test suite bug fixes

commit 5dfaf627bb2f6566cdb9293e5941be51afaad508
Author: Bayard <bayardneville@gmail.com>
Date:   Tue Dec 20 22:57:06 2016 -0500

    fixes

commit 428c6fcba7cc49e71e98390949469bef52bc4d1b
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Tue Dec 20 22:46:56 2016 -0500

    test script bug fix

commit ce499daaf16222d9b62c8256a5abd6ba8fbf9041
Author: Bayard <bayardneville@gmail.com>
Date:   Tue Dec 20 21:13:09 2016 -0500

    fixed more tests

commit 065c012310005b488bce6dd42e3989c5130a2391
Merge: eb691f3 caffed4
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Tue Dec 20 20:45:32 2016 -0500

    Merge branch 'master' of https://github.com/bayardneville/goblin

commit eb691f3363f488f78a0fcbcd97f417cbe81dc69e
Author: Gabriel Uribe <gu2124@columbia.edu>
```

Date: Tue Dec 20 20:45:28 2016 -0500

    more tests

commit caffed4e44ec5977965fcd135b24df93374b2868
Author: Bayard <bayardneville@gmail.com>
Date: Tue Dec 20 20:41:19 2016 -0500

    fixed some tests

commit eb1aaf27c3323af8bd1408faaee8956fc431a792
Author: Bayard <bayardneville@gmail.com>
Date: Tue Dec 20 20:32:13 2016 -0500

    bug fixes

commit 96cd0e4285fd4cabd1a4e8c01e86725c81a82480
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Tue Dec 20 20:02:24 2016 -0500

    first batch of fail tests

commit 2fb6f5617cf42e2ba0bdaa8f17e64051ff1d38a9
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Tue Dec 20 19:40:10 2016 -0500

    Gobble anywhere! be free

commit 33a586612fe3896e24b62eb816c593f26182277c
Author: Bayard <bayardneville@gmail.com>
Date: Tue Dec 20 19:29:56 2016 -0500

    added test output generator

commit a24f2a25a089420fd8693d272ba80a34c5993fa1
Author: Bayard <bayardneville@gmail.com>
Date: Tue Dec 20 17:48:49 2016 -0500

    updated goblin hunter

commit 515c4f7a8300c94a0b7843cfe8b0c40855a9a4fe
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Tue Dec 20 17:12:02 2016 -0500

```
    gobble update

commit d7a3d22231de7ea9ef136aade23012e19731f873
Merge: cdb7184 fba4f6c
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Tue Dec 20 17:01:22 2016 -0500

    Merge branch 'master' of https://github.com/bayardneville/goblin

commit cdb7184cd0a34728fa831f77b432231cd63b6f48
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Tue Dec 20 17:01:18 2016 -0500

    gobble file update, c file updates

commit fba4f6c532ba7b61b36732de0315d81819221aed
Author: Bayard <bayardneville@gmail.com>
Date:   Tue Dec 20 16:55:36 2016 -0500

    removed accidentally committed binary file

commit 07eda5da88c0149fef1ef78e4ee04eae92d5b29c
Author: Bayard <bayardneville@gmail.com>
Date:   Tue Dec 20 16:54:19 2016 -0500

    Fixed some documentation

commit 6dd75986994dd3f21bb670ffd5817f001e7c106f
Merge: 11e5362 14f6ef1
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Tue Dec 20 16:42:48 2016 -0500

    Merge branch 'master' of https://github.com/bayardneville/goblin

commit 11e536254ad3a70c00aef3231cf783bce7447d90
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Tue Dec 20 16:42:45 2016 -0500

    test script updated, new tests working, added test skel code, updated gitignore

commit 14f6ef10ba97111d1beecddc3071288850888e98
Author: Christina Floristean <Christina@dyn-209-2-211-240.dyn.columbia.edu>
```

Date: Tue Dec 20 16:37:53 2016 -0500

    game change

commit 3f62eaa026d9307f70ee2d90da56770a4fac7a2f
Author: Christina Floristean <Christina@dyn-209-2-211-240.dyn.columbia.edu>
Date: Tue Dec 20 16:14:00 2016 -0500

    game change

commit 4c7fc9915e6c03a90be9a19ad78b54c8c7a4c68f
Author: Christina Floristean <Christina@dyn-209-2-211-240.dyn.columbia.edu>
Date: Tue Dec 20 15:52:50 2016 -0500

    Made goblins move twice in one round instead.

commit 7aa1ee3685f706cb1553160e7df131207b69046f
Author: Christina Floristean <Christina@dyn-209-2-211-240.dyn.columbia.edu>
Date: Tue Dec 20 15:40:24 2016 -0500

    Made goblins move every other round.

commit be2b32a52ae9354bef4e22f70b39745ee0445909
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Tue Dec 20 14:58:47 2016 -0500

    Reenables echo bit in terminal after game ends

commit 17b233a2dc316a5d393305f524ce57b55c1a1d80
Author: Christina Floristean <Christina@dyn-209-2-210-104.dyn.columbia.edu>
Date: Tue Dec 20 12:38:45 2016 -0500

    Added mod to semant, created goblin_hunter test program

commit 45983e26f59436eb63f974c2b6339690e81c19b6
Author: Bayard <bayardneville@gmail.com>
Date: Tue Dec 20 01:09:59 2016 -0500

    fixed module name

commit c66114309246a618f7af981cc28ee4f18f5d58b5
Author: Bayard <bayardneville@gmail.com>
Date: Tue Dec 20 00:29:33 2016 -0500

made helloworld status stay on screen

commit c64631e973cc51a8857efa45c347854468657ce0
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Tue Dec 20 00:00:00 2016 -0500

Fixed control keys echoing to game screen

commit a4b39c7fb3001aef0fc1d58aa1c0af6ba3ae8fc3
Author: Bayard <bayardneville@gmail.com>
Date: Mon Dec 19 22:20:37 2016 -0500

made helloworld script use gobbler

commit 6b2e56bb542ec89eb3ee0e6ec48aa93532092ed2
Author: Bayard <bayardneville@gmail.com>
Date: Mon Dec 19 18:36:23 2016 -0500

redesigned world intialization

commit 052bca0cbe1d99ba2bbbf8d93b7317fad1d5aed2
Author: Bayard <bayardneville@gmail.com>
Date: Mon Dec 19 17:29:20 2016 -0500

fixed semantic checking

commit b8c9dc63a24583b63dafec105889f65b42d4c9b2
Author: Christina Floristean <Christina@dyn-209-2-210-104.dyn.columbia.edu>
Date: Mon Dec 19 14:52:06 2016 -0500

Semantic checking w/o place and prints.

commit 3b4d81d629bbd9752b430a8be6e7c9e2b4eb4e05
Author: kevinkxiao <kaifanxiao@gmail.com>
Date: Sun Dec 18 19:00:40 2016 -0500

Added some documentation

commit cb245c74554022c676fced488546603959604fcb
Author: Bayard <bayardneville@gmail.com>
Date: Sun Dec 18 18:37:44 2016 -0500

```
    fixed strings in scanner

commit 58c03a9b5dc58b4ea611da3c4ce2c8e234581c11
Merge: e14f48c 8e5d841
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Sun Dec 18 17:58:45 2016 -0500

    Merge branch 'master' of https://github.com/bayardneville/goblin

commit e14f48c026cb01a72ba31239a8b12e9ef9d878b2
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Sun Dec 18 17:56:38 2016 -0500

    Updated gobble script

commit c2c18b0268488dab92a7c9b8c89cec0ad5edb82e
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Sun Dec 18 17:56:20 2016 -0500

    Updated gobble script

commit 8e5d841d61f22a14d0ab54e882a4a5553085a7e8
Author: Bayard <bayardneville@gmail.com>
Date:   Sun Dec 18 17:48:36 2016 -0500

    added char type, example program

commit 11918ad89405738f383161859113b2b5d6b7afd0
Author: Bayard <bayardneville@gmail.com>
Date:   Sun Dec 18 16:49:12 2016 -0500

    added type checking, improved game loop, fixed several warnings

commit 39fc14e63e1ff257980d61592a705a592480de33
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Sun Dec 18 16:39:55 2016 -0500

    Added gobble linker script

commit efb954c86eba864c1a6828775ef69a96d92504d4
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Sun Dec 18 15:28:53 2016 -0500
```

    Clear screen function

commit 79b431f10070454815014f3181a306fad93fcc21
Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 17 23:05:54 2016 -0500

    fixed tester somewhat

commit a618fb39648a62eca24537ead193e5b071acea94
Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 17 20:15:36 2016 -0500

    linked getKey

commit 4ee1b16ac311b0006a5daac1f169409d96c517d7
Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 17 19:08:35 2016 -0500

    improved game loop ordering and basic demo program

commit 299335c6b4ab219869dee96429d03d76baa32cc5
Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 17 18:44:10 2016 -0500

    added game loop until exit is encountered, board init still must occur in test method

commit f67cf003062a899f6bf3165a79634793139945d9
Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 17 18:10:16 2016 -0500

    added removing of entities from board

commit 5a103fa89e4a74c98c8f3f30373ed11a97d5b3b9
Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 17 17:58:44 2016 -0500

    added move method and fixed does call on empty tile

commit 70951e07566aca5ad24da17000106021257fb710
Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 17 17:06:59 2016 -0500

    implemented build and does

```
commit 22b1375da951d054a4513da3d0cef67dbb736604
Author: Bayard <bayardneville@gmail.com>
Date:   Fri Dec 16 21:56:11 2016 -0500

    added board printing and fixed field access bug

commit 655b3c08bc950f2ad5e901a762794a248ade2a4f
Author: Bayard <bayardneville@gmail.com>
Date:   Fri Dec 16 21:17:57 2016 -0500

    fixed type code indexes and removed unused string type

commit 3ec022844b530bd8452bc0077e9deb4288d07830
Author: Bayard <bayardneville@gmail.com>
Date:   Fri Dec 16 20:43:09 2016 -0500

    fixed field access

commit 78adda62ca08d5920023b78371cb19c2abb1853a
Author: Bayard <bayardneville@gmail.com>
Date:   Fri Dec 16 14:30:36 2016 -0500

    added user defined structs, placing structs on board, field access, and field assignment

commit 820e37a506a5a6dd4aeca48677737d60e12d7278
Author: Bayard <bayardneville@gmail.com>
Date:   Thu Dec 15 19:59:06 2016 -0500

    fixed helloworld

commit e8a9769ab9d32ac8855fd4d6ade9dfa2e486cd3e
Merge: 9a560f2 081d7f9
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Thu Dec 15 19:57:28 2016 -0500

    Merge branch 'master' of https://github.com/bayardneville/goblin

commit 9a560f2a65708ee4ec14ee7097ec15db8652892d
Author: Gabriel Uribe <gu2124@columbia.edu>
Date:   Thu Dec 15 19:57:04 2016 -0500

    renamed microc
```

commit 6a97ed8c46a4d3794e9664d7ae30e4edb4684400
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Thu Dec 15 19:56:14 2016 -0500

    updated Makefile names

commit 081d7f96e52ea52d835c77e06e5c19759468586f
Author: Bayard <bayardneville@gmail.com>
Date: Thu Dec 15 19:49:48 2016 -0500

    translated worlds and entities for structs, added entity to symbol mapping list

commit a0418e0eb01867aa549df1422008638a37695ccd
Author: Bayard <bayardneville@gmail.com>
Date: Thu Dec 15 19:08:51 2016 -0500

    made it possible to iterate over the board and access tiles and their fields

commit f5f8480bf7acd171d15ef1831afbf290515593c5
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Thu Dec 15 18:51:28 2016 -0500

    c function for key input added

commit c3fbe1e0a3092ccf04e15997b02eb5ab0d26f7fc
Author: Bayard <bayardneville@gmail.com>
Date: Thu Dec 15 13:53:08 2016 -0500

    added x and y fields to tiles and refactored tile access functions

commit 526e87bee890558ed85289b8036847cb36a6a64d
Author: Bayard <bayardneville@gmail.com>
Date: Wed Dec 14 18:02:52 2016 -0500

    started basic place function

commit d5c93bfbc93168bd2dd43175e7735135b5fa4799
Author: Bayard <bayardneville@gmail.com>
Date: Wed Dec 14 17:44:48 2016 -0500

    added tile struct, basic struct access, variable board size

commit 096c09c14ec7f0970310c662f7e3912f2b290ff5
Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 10 20:37:21 2016 -0500

    conversion for build and does to functions

commit 1f6a1e976f35782705965a61edcf6b130a79ce34
Author: Bayard <bayardneville@gmail.com>
Date: Fri Dec 9 00:51:44 2016 -0500

    made function named test required and fixed some tests

commit 689064c0aca803ad26671e2376e26f620f4dbfa9
Author: Bayard <bayardneville@gmail.com>
Date: Thu Dec 8 23:38:17 2016 -0500

    added basic arrays and peekAt

commit 454bd08939d95b2e3ef5ba1d5ce92b8f85c47919
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Wed Dec 7 17:49:08 2016 -0500

    fixed return statements

commit f6aa7ce5293600c9650c77db6ff60e3cb3feaed7
Author: Bayard <bayardneville@gmail.com>
Date: Tue Dec 6 22:12:05 2016 -0500

    started implementing arrays

commit 9549b5cb71af3afc1ba3c5dfbe1b62920592fe92
Author: Bayard <bayardneville@gmail.com>
Date: Tue Dec 6 21:00:38 2016 -0500

    removed redundant sast file

commit 10b856f76d1efa1dff324bd66860ebdc4a5b66f8
Author: Bayard <bayardneville@gmail.com>
Date: Tue Dec 6 20:49:51 2016 -0500

    changed worlds in parser to include board array as field

commit afde014712b8399e5355b273e00e2ed0ec6ba889

Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 3 17:59:59 2016 -0500

    basic game loop

commit caddf1a54d4283fe450ae48289789489613fa965
Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 3 16:07:01 2016 -0500

    added basic sast

commit c1b159540a2c8a415cab3fe7843c9f33c4ad46a6
Author: Bayard <bayardneville@gmail.com>
Date: Sat Dec 3 15:55:12 2016 -0500

    put globals back into codgen

commit 6970e556122af4c58b0b3b5d6d964ec016cfbf3e
Author: Bayard <bayardneville@gmail.com>
Date: Thu Dec 1 22:16:29 2016 -0500

    made tester produce individual output files

commit 7e19a30d7b49eded667824a2ff0e57a66c091333
Author: Bayard <bayardneville@gmail.com>
Date: Thu Dec 1 21:56:09 2016 -0500

    updated some tests

commit 7850a168a09241a59cb8779ec588ce2b8a6ad0a7
Author: Bayard <bayardneville@gmail.com>
Date: Thu Dec 1 21:38:09 2016 -0500

    fixed gitignore and added missing files

commit 09616313cea0e91b701ca8754227afbb9690790d
Author: Bayard <bayardneville@gmail.com>
Date: Thu Dec 1 21:35:11 2016 -0500

    fixed another test script bug

commit b5367405fcf17446534737914bc9cdbbf704f865
Author: Bayard <bayardneville@gmail.com>

Date: Thu Dec 1 21:18:33 2016 -0500

    fixed bugs in test script and renamed test files

commit 4d614cbbe0047097d699eaaea9c145ca58448f06
Author: Bayard <bayardneville@gmail.com>
Date: Tue Nov 29 21:44:19 2016 -0500

    made make clean remove more file types

commit f923209cea0ae49a728322391430609adb1ccef6
Merge: cff0708 c734174
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Mon Nov 21 02:39:12 2016 -0500

    Merge branch 'master' of https://github.com/bayardneville/goblin

commit cff070843b31e341d57d26f8becc116fec22bd59
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Mon Nov 21 02:38:59 2016 -0500

    new testing script

commit c7341747e8851f716f6ad3250b1bc252d80cc9a8
Author: Bayard <bayardneville@gmail.com>
Date: Sat Nov 19 17:13:26 2016 -0500

    fixed strings in scanner and wrote hello world test

commit e0409db533551ddf2e2bc296a36a75b9014ce8af
Author: Bayard <bayardneville@gmail.com>
Date: Sat Nov 19 17:03:55 2016 -0500

    added some types and handling of ast nodes to codegen

commit 994f63d725b5b3a8724f4c62476b5c8ddf6fb8c7
Author: Bayard <bayardneville@gmail.com>
Date: Sat Nov 19 16:34:12 2016 -0500

    made build and does blocks be optionals instead of lists

commit 3a92a6fa344211aed9949c668ec28d553119ff33
Author: Bayard <bayardneville@gmail.com>

Date: Sat Nov 19 15:50:53 2016 -0500

    removed helloworld binary and updated script

commit 98a5f1cc3885c4985949befb87e00f6b73ad20fe
Author: Bayard <bayardneville@gmail.com>
Date: Thu Nov 17 22:42:21 2016 -0500

    got string printing working and added basic hello world script

commit 5d516d7b7cca29de6dcca72048bd2811d4229e97
Author: Christina Floristean <Christina@dyn-209-2-210-59.dyn.columbia.edu>
Date: Sat Nov 12 19:58:38 2016 -0500

    Changed variable initialization in parser.

commit 12c030aacc563c33d963083aad1db13bb24894be
Author: Christina Floristean <Christina@dyn-209-2-210-59.dyn.columbia.edu>
Date: Sat Nov 12 19:48:48 2016 -0500

    Changed printing on ast.

commit a87e6c33f841c99f145ae9eeb8bf38f92b11f8d9
Author: Bayard <bayardneville@gmail.com>
Date: Sat Nov 12 19:44:51 2016 -0500

    fixed dot fiel access precedence issue

commit 556bec05676ecb030b2169be5e0f56a7b4388583
Author: Christina Floristean <Christina@dyn-209-2-210-59.dyn.columbia.edu>
Date: Sat Nov 12 17:45:49 2016 -0500

    Added TypeCheck to ast.

commit c294cefa08e92887b02e6bd334066d173e415adb
Author: Bayard <bayardneville@gmail.com>
Date: Sat Nov 12 17:34:04 2016 -0500

    made is and instanceof not be Binops

commit 2fc1d2384444a00cefed80c3b76f2c0a7e29dcb3
Merge: 0afc80f 43eec2e
Author: Christina Floristean <Christina@dyn-209-2-210-59.dyn.columbia.edu>

Date: Sat Nov 12 16:28:45 2016 -0500

    Added pretty printing to ast and changed dot operator in parser.

commit 0afc80fd930c4a5fe35d8a64ae2757ce951977a0
Author: Christina Floristean <Christina@dyn-209-2-210-59.dyn.columbia.edu>
Date: Sat Nov 12 16:24:57 2016 -0500

    Added pretty printing to ast and changed dot operator in parser.

commit 43eec2e3896a3b6ce458e698e6bc822c4cafa907
Author: Bayard <bayardneville@gmail.com>
Date: Sat Nov 12 15:33:50 2016 -0500

    made test script not require updating path

commit 36a44d5877e19465b0e97564a7905b8b56ab5763
Author: kevinkxiao <kaifanxiao@gmail.com>
Date: Sat Nov 12 15:26:46 2016 -0500

    Realigned confusing formatting

commit e16b2d4a7e03244f895ac7c4927ddadb54e680a8
Merge: efb0010 4ff9bf7
Author: kevinkxiao <kaifanxiao@gmail.com>
Date: Sat Nov 12 15:24:34 2016 -0500

    Merge branch 'master' of https://github.com/bayardneville/goblin

commit 4ff9bf79cfe5c29d58cb355e2ca62ad8f07e82ed
Merge: ef3dacf 69e7c0b
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Sat Nov 12 14:50:19 2016 -0500

    Merge branch 'master' of https://github.com/bayardneville/goblin

commit ef3dacf2a82568ecf24e69dfd942ac3194160d2f
Author: Gabriel Uribe <gu2124@columbia.edu>
Date: Sat Nov 12 14:49:50 2016 -0500

    ast

commit 69e7c0b0ede47bb1745930f373c8d3e778a5d215

Author: Bayard <bayardneville@gmail.com>
Date: Fri Nov 11 18:05:06 2016 -0500

    finished removing void from parser

commit 90c75d1e04f6643b6d81c6811ffe7da9eab4deeb
Author: Bayard <bayardneville@gmail.com>
Date: Fri Nov 11 17:05:39 2016 -0500

    added instanceof operator

commit f7809442d1adfbf03e008210e9890ca9ffc7b128
Author: Bayard <bayardneville@gmail.com>
Date: Sun Nov 6 22:37:37 2016 -0500

    removed void and specific entity types

commit 9529b7c096f0c4b239529d5064fbe57898fb79b3
Author: Bayard <bayardneville@gmail.com>
Date: Sun Nov 6 22:10:09 2016 -0500

    added is for type checking

commit 7a77865c1c056fcb12a26592446b932be93167cd
Author: Bayard <bayardneville@gmail.com>
Date: Sun Nov 6 20:41:21 2016 -0500

    removed controls block and distinct player def

commit efb0010fa107453e77975c6dfee78f71453e26e2
Author: Bayard <bayardneville@gmail.com>
Date: Sun Nov 6 20:41:21 2016 -0500

    added modulo operator

commit cae9bd11b1475b86f8ef64af29bdb0ba3b2beeed
Author: Bayard <bayardneville@gmail.com>
Date: Sun Nov 6 20:29:38 2016 -0500

    made user defined entities possible as variable and return types

commit 8ef7db64c6fc202df7959eaff1b8332b2d3096ae
Author: Bayard <bayardneville@gmail.com>

Date: Sun Nov 6 19:31:39 2016 -0500

    added basic .gitignore

commit cb4e45c3d2bf664b50cafc32b56535099a0db41e
Author: Bayard <bayardneville@gmail.com>
Date: Tue Nov 1 22:25:18 2016 -0400

    added inheritance

commit 65ea844b60146d0d4f37b797ba623e9bbd31d3fb
Author: Bayard <bayardneville@gmail.com>
Date: Tue Nov 1 22:00:47 2016 -0400

    refactored world

commit 4ed73d40b1c127be21afaac8bcac940e5472dca2
Author: Bayard <bayardneville@gmail.com>
Date: Tue Nov 1 21:53:39 2016 -0400

    changed typ to ftyp and reversed build and does bodies

commit 3c8bfa51a00017b8d10654145c2577b96b25d67d
Author: Bayard <bayardneville@gmail.com>
Date: Tue Nov 1 21:46:20 2016 -0400

    refactored declarations

commit 80dd28bedfdf823d80e31ac8ba54d0c1388e143a
Author: Bayard <bayardneville@gmail.com>
Date: Tue Nov 1 21:21:06 2016 -0400

    removed coord

commit 9ecaa309b03e028e5bf0cdfe83ef707fc6ba2455
Author: Bayard <bayardneville@gmail.com>
Date: Tue Nov 1 20:35:25 2016 -0400

    changed functions to behaviors

commit 5b1c06b79edfd3e138fe05d73491a144c37d073c
Author: Bayard <bayardneville@gmail.com>
Date: Tue Nov 1 20:18:08 2016 -0400

```
    added +=,*=, etc.

commit 1aef9e7df6dcd71fc56277857291d5ef5054ec83
Author: Bayard <bayardneville@gmail.com>
Date:   Tue Oct 25 20:46:56 2016 -0400

    removed floats since they only add confusion

commit 03a7905a7610737bf2c3af4e1f73dc5807115a64
Author: Bayard <bayardneville@gmail.com>
Date:   Sun Oct 23 23:29:46 2016 -0400

    added dot notation field access

commit c56a9242af651f157d93c97adb145949cc8c487a
Author: Bayard <bayardneville@gmail.com>
Date:   Sun Oct 23 23:06:20 2016 -0400

    added coord type

commit e8f943581e7fffbf267b4bc726fb3abaa59f8a7c
Author: Bayard <bayardneville@gmail.com>
Date:   Sun Oct 23 21:45:34 2016 -0400

    removed square brackets from the language as they are unused

commit bc010296bbafdcb93b9b16ac71835198316e7d62
Author: Bayard <bayardneville@gmail.com>
Date:   Sun Oct 23 21:28:30 2016 -0400

    made variable declarations require an initial value

commit 2b24e561e671bdd7bde0356e4ea650e4c0b3906a
Author: Gabriel Uribe <gabe@dyn-160-39-234-244.dyn.columbia.edu>
Date:   Fri Oct 21 19:01:44 2016 -0400

    fixed argument OBOE

commit 56bc9f008175b0f51dd89d23852acbe877d54a46
Author: Gabriel Uribe <gabe@dyn-160-39-234-244.dyn.columbia.edu>
Date:   Fri Oct 21 18:57:38 2016 -0400
```

changed does block and added control block

commit 871454a66f15314d5d952bb8979ac3cb79149b91
Author: Bayard <bayardneville@gmail.com>
Date: Sat Oct 15 21:29:27 2016 -0400

added entity declarations

commit a8a277349aa731d381903324bf5a791e99721501
Author: Bayard <bayardneville@gmail.com>
Date: Sat Oct 15 21:14:05 2016 -0400

changed player to dict

commit 5aa4902f8c59e2acb690f47af80e1994872b26c0
Author: Christina Floristean <Christina@Christinas-MacBook-Air.local>
Date: Sat Oct 15 21:00:00 2016 -0400

Changed keymap

commit a4ee2d906d847a68f1c5136b4d182456f8a33998
Author: Bayard <bayardneville@gmail.com>
Date: Sat Oct 15 20:55:51 2016 -0400

player keymappings

commit 9038fa0412fd8e2409c0bad07d06023620a1a427
Author: Bayard <bayardneville@gmail.com>
Date: Sat Oct 15 20:10:31 2016 -0400

continued section implementation

commit 6613037d43ab59acad20351e428ce3588e1d076f
Author: Bayard <bayardneville@gmail.com>
Date: Sat Oct 15 19:28:08 2016 -0400

started implementing sections

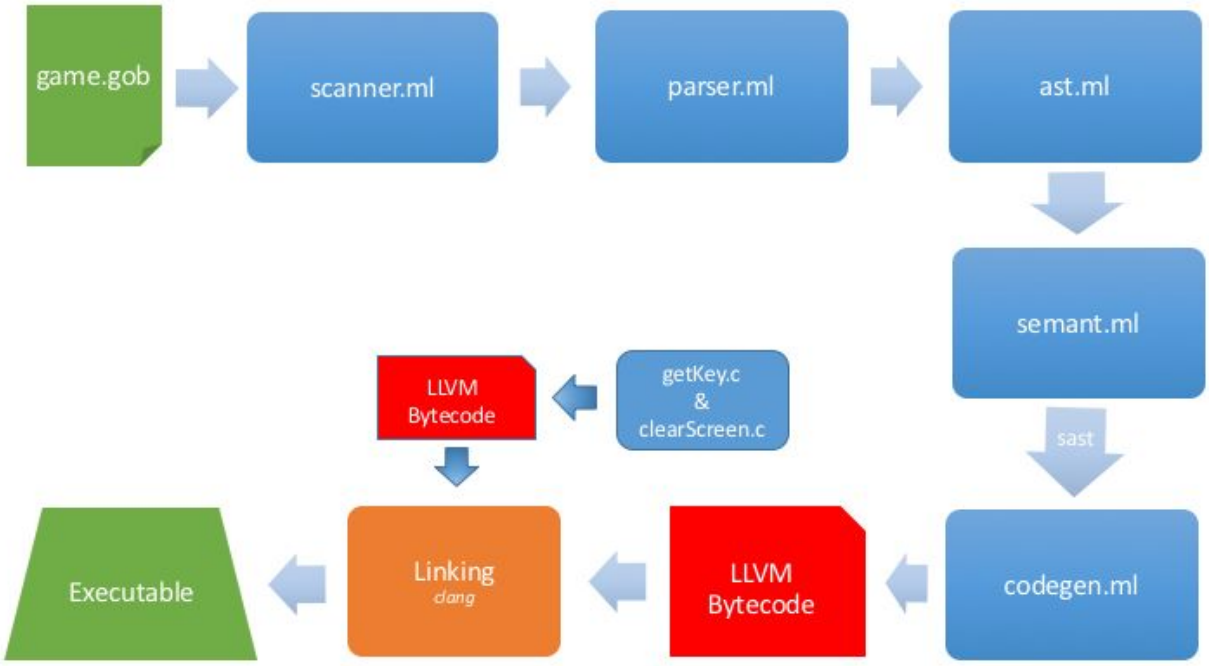
commit 5df85e682f0abef5adf401f9889b6926eb272924
Author: Bayard <bayardneville@gmail.com>
Date: Sat Oct 15 17:18:32 2016 -0400

added strings and floats to AST

```
commit 33a27b4c18a31002444ae1813c8314173bafaac2
Author: Bayard <bayardneville@gmail.com>
Date:   Wed Oct 12 22:20:18 2016 -0400

    added floats and strings

commit 541ed02a4770b95649ff05d1880f17454406ae90
Author: Bayard <bayardneville@gmail.com>
Date:   Wed Oct 12 21:53:45 2016 -0400

    added strings to scanner and new tokens to parser

commit e3b08db2e70156ce5073e0d88c93c2006c299272
Author: Bayard <bayardneville@gmail.com>
Date:   Wed Oct 12 21:25:06 2016 -0400

    modified scanner to match goblin syntax

commit 39bbc7b4553e2e0a8151209252274020ee7b2a4c
Author: Bayard <bayardneville@gmail.com>
Date:   Wed Oct 12 20:53:52 2016 -0400

    added microc baseline
```

5. Architectural Design

Translator Diagram



Interfaces between Components

The gobble compiler for goblin takes in a single goblin source program, typically a .gob file but there is no requirements on file endings, and outputs an executable.

The code goes through the following steps inside the gobble compiler: The scanner, defined in scanner.ml, takes in the source file and produces a stream of tokens. This stream of tokens is then passed to the parser, which creates a representation of the program as an abstract syntax tree (AST), with three key pieces: the world initialization, the entities, and the functions. The nodes of this tree are defined in ast.ml. The parser then passes the AST to the semantic checker, which processes the AST into a more C-like form that codegen understands and checks to make sure the program is semantically valid. This version of the AST that the codegen understands is the SAST. It consists of global variables, functions, struct definitions, and symbol pairings. It adds a global variable for the game board, as well as for the number of rows and columns in the game board. It also add all of the “build” and “does” blocks to the function list from the AST. The display symbols for the entities are also put into a list of pairs of entity names and symbols. The main game loop is also added to the functions list at this point. The SAST is then checked for semantic correctness and passed to the codegen. The codegen, defined in codegen.ml, reads the SAST and generates corresponding LLVM code that represents the goblin program. This LLVM file generated by the compiler must then be linked

with external C dependencies needed for getting user input and printing to the screen and then compiled to machine code. The linking and final compilation from LLVM IR code is done using clang.

Contributors

We all implemented the scanner, parser, and AST together. Christina implemented the semantic checker. Bayard and Kevin implemented the semantic analyzer and codegen. Gabe implemented the external C libraries and linker.

6. Test Plan

Source Program 1: Goblin Hunter

```
world[20,60] {
    num i;
    num j;
    for(i = 0; i < rows; i += rows - 1) {
      for(j = 0; j < cols; j += 1) {
        place(wall, i, j);
      }
    }
    for(i = 0; i < rows; i += 1) {
      for(j = 0; j < cols; j += cols - 1) {
        place(wall, i, j);
      }
    }
    place(goblin, 5, 10);
    place(goblin, 15, 35);
    place(goblin, 16, 15);
    place(trap, 11, 35);
    place(trap, 15, 20);
    place(trap, 5, 5);
    place(player, 5, 5);
}
entities {
  @:player {
    num attack;
    num lives_left;
    build {
      this.attack = 5;
      this.lives_left = 3;
    }
```

```
    does {
      playerMove(this);
    }
  }
  g:goblin {
    num health;
    num k;
    build {
      this.health = 5;
      this.k = 0;
    }
    does {
      goblinMove(this);
    }
  }
  T:trap {}
  #:wall {}
}
functions {
  bool playerMove(entity p) {
    num r;
    num c;
    char k;
    bool moved;
    moved = false;
    r = row(p);
    c = col(p);
    prints("Press wasd to move or q to quit");
    while(not moved) {
      k = getKey();
      if(k == 'w' and r > 0) {
        if(moveOrAttack(p, r - 1, c)) {
          moved = true;
        }
      } else if(k == 'a' and c > 0) {
        if(moveOrAttack(p, r, c - 1)) {
          moved = true;
        }
      } else if(k == 's' and r < rows - 1) {
        if(moveOrAttack(p, r + 1, c)) {
          moved = true;
        }
      } else if(k == 'd' and r < cols - 1) {
```

```
      if(moveOrAttack(p, r, c + 1)) {
        moved = true;
      }
    } else if(k == 'q') {
      exit;
      moved = true;
    }
  }
  return true;
}

bool goblinMove(entity p) {
  num r;
  num c;
  num k;
  r = row(p);
  c = col(p);
  if(r % 2 == 1) {
    k = 1;
  }
  if(k == 0) {
    moveIfEmpty(p, r - 1, c);
    k = 1;
  } else if(k == 1) {
    moveIfEmpty(p, r, c - 1);
    k = 2;
  } else if(k == 2) {
    moveIfEmpty(p, r + 1, c );
    k = 3;
  } else if(k == 3) {
    moveIfEmpty(p, r, c + 1);
    k = 4;
  } else if(k == 4) {
    k = 0;
  }
  return true;
}

bool moveIfEmpty(entity p, num r, num c) {
  entity e;
  e = peek(r, c);
  if(e is empty) {
     move(p, r, c);
```

```
      return true;
    }
    return false;
  }

  bool moveOrAttack(entity p, num r, num c) {
    entity e;
    e = peek(r, c);
    if(e is goblin) {
      e.health -= p.attack;
      prints("dealt 5 damage to goblin");
      if(e.health <= 0) {
        prints("killed goblin! press any key to continue");
        getKey();
        remove(e);
        move(p, r, c);
      }
    }
    else if(e is trap) {
      prints("stepped on a trap! you are dead!");
      exit;
    }
    else if (e is empty) {
       move(p, r, c);
    } else {
      return false;
    }
    return true;
  }
}
```

LLVM Program 1: Goblin Hunter

```
; ModuleID = 'Goblin'

%_tile = type { i32, i32, i32, i8* }
%wall = type { i1 }
%goblin = type { i1, i32, i32 }
%trap = type { i1 }
%player = type { i1, i32, i32 }

@_board = global [20 x [60 x %_tile]] zeroinitializer
@rows = global i32 0
```

```
@cols = global i32 0
@_exit = global i1 false
@_symbols = global [5 x i8] c" #Tg@"
@_build_tbl = global [5 x i1 (%_tile*)*] [i1 (%_tile*)* @_nothing, i1 (%_tile*)* @_build_wall, i1
(%_tile*)* @_build_trap, i1 (%_tile*)* @_build_goblin, i1 (%_tile*)* @_build_player]
@_does_tbl = global [5 x i1 (%_tile*)*] [i1 (%_tile*)* @_nothing, i1 (%_tile*)* @_does_wall, i1
(%_tile*)* @_does_trap, i1 (%_tile*)* @_does_goblin, i1 (%_tile*)* @_does_player]
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [3 x i8] c"%c\00"
@.str = private unnamed_addr constant [2 x i8] c"\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.3 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.5 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.6 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.7 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.8 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.9 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.10 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.11 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.12 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.13 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.14 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.15 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.16 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.17 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.18 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.19 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.20 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.21 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.22 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.23 = private unnamed_addr constant [3 x i8] c"%c\00"
@.str.24 = private unnamed_addr constant [26 x i8] c"dealt 5 damage to goblin\0A\00"
@.str.25 = private unnamed_addr constant [42 x i8] c"killed goblin! press any key to
continue\0A\00"
@.str.26 = private unnamed_addr constant [34 x i8] c"stepped on a trap! you are dead!\0A\00"
@fmt.27 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.28 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.29 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.30 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.31 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.32 = private unnamed_addr constant [3 x i8] c"%c\00"
@.str.33 = private unnamed_addr constant [33 x i8] c"Press wasd to move or q to quit\0A\00"
```

```
declare i32 @printf(i8*, ...)

declare i8 @getKey()

declare i32 @_clearScreen()

define i32 @main() {
entry:
  %i = alloca i32
  %j = alloca i32
  %e = alloca %_tile*
  store i32 20, i32* @rows
  store i32 60, i32* @cols
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %merge, %entry
  %i11 = load i32, i32* %i
  %rows = load i32, i32* @rows
  %tmp12 = icmp slt i32 %i11, %rows
  br i1 %tmp12, label %while_body, label %merge13

while_body:                                       ; preds = %while
  store i32 0, i32* %j
  br label %while1

while1:                                           ; preds = %while_body2, %while_body
  %j7 = load i32, i32* %j
  %cols = load i32, i32* @cols
  %tmp8 = icmp slt i32 %j7, %cols
  br i1 %tmp8, label %while_body2, label %merge

while_body2:                                      ; preds = %while1
  %i3 = load i32, i32* %i
  %j4 = load i32, i32* %j
  %_board = getelementptr [20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i1 false, i32 %i3,
i32 %j4
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %_board, i32 0, i32 0
  store i32 %i3, i32* %_tile_fld_ptr
  %_tile_fld_ptr5 = getelementptr inbounds %_tile, %_tile* %_board, i32 0, i32 1
  store i32 %j4, i32* %_tile_fld_ptr5
  %j6 = load i32, i32* %j
```

```
  %tmp = add i32 %j6, 1
  store i32 %tmp, i32* %j
  br label %while1

merge:                                            ; preds = %while1
  %i9 = load i32, i32* %i
  %tmp10 = add i32 %i9, 1
  store i32 %tmp10, i32* %i
  br label %while

merge13:                                          ; preds = %while
  %_init_world_result = call i1 @_init_world()
  br label %while14

while14:                                          ; preds = %merge65, %merge13
  %_exit = load i1, i1* @_exit
  %tmp66 = xor i1 %_exit, true
  br i1 %tmp66, label %while_body15, label %merge67

while_body15:                                     ; preds = %while14
  %_clearScreen = call i32 @_clearScreen()
  store i32 0, i32* %i
  br label %while16

while16:                                          ; preds = %merge36, %while_body15
  %i40 = load i32, i32* %i
  %rows41 = load i32, i32* @rows
  %tmp42 = icmp slt i32 %i40, %rows41
  br i1 %tmp42, label %while_body17, label %merge43

while_body17:                                     ; preds = %while16
  store i32 0, i32* %j
  br label %while18

while18:                                          ; preds = %merge26, %while_body17
  %j33 = load i32, i32* %j
  %cols34 = load i32, i32* @cols
  %tmp35 = icmp slt i32 %j33, %cols34
  br i1 %tmp35, label %while_body19, label %merge36

while_body19:                                     ; preds = %while18
  %i20 = load i32, i32* %i
  %j21 = load i32, i32* %j
```

```
  %_board22 = getelementptr [20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9 0, i32 %i20,
i32 %j21
  store %_tile* %_board22, %_tile** %e
  %e23 = load %_tile*, %_tile** %e
  %_tile_fld_ptr24 = getelementptr inbounds %_tile, %_tile* %e23, i32 0, i32 2
  %_e_type_ptr = load i32, i32* %_tile_fld_ptr24
  %_tmp = icmp eq i32 %_e_type_ptr, 0
  %tmp25 = xor i1 %_tmp, true
  br i1 %tmp25, label %then, label %else

merge26:                                          ; preds = %else, %then
  %e29 = load %_tile*, %_tile** %e
  %_tile_fld_ptr30 = getelementptr inbounds %_tile, %_tile* %e29, i32 0, i32 2
  %_tcode = load i32, i32* %_tile_fld_ptr30
  %_sym_ptr = getelementptr [5 x i8], [5 x i8]* @_symbols, i8 0, i32 %_tcode
  %_sym = load i8, i8* %_sym_ptr
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt.1, i32 0,
i32 0), i8 %_sym)
  %j31 = load i32, i32* %j
  %tmp32 = add i32 %j31, 1
  store i32 %tmp32, i32* %j
  br label %while18

then:                                             ; preds = %while_body19
  %e27 = load %_tile*, %_tile** %e
  %_tile_fld_ptr28 = getelementptr inbounds %_tile, %_tile* %e27, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr28
  %_ent_ptr = bitcast i8* %_void_ptr to i1*
  store i1 false, i1* %_ent_ptr
  br label %merge26

else:                                             ; preds = %while_body19
  br label %merge26

merge36:                                          ; preds = %while18
  %printf37 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8], [2 x i8]* @.str, i32 0,
i32 0))
  %i38 = load i32, i32* %i
  %tmp39 = add i32 %i38, 1
  store i32 %tmp39, i32* %i
  br label %while16

merge43:                                          ; preds = %while16
```

```
  store i32 0, i32* %i
  br label %while44

while44:                                          ; preds = %merge59, %merge43
  %i62 = load i32, i32* %i
  %rows63 = load i32, i32* @rows
  %tmp64 = icmp slt i32 %i62, %rows63
  br i1 %tmp64, label %while_body45, label %merge65

while_body45:                                     ; preds = %while44
  store i32 0, i32* %j
  br label %while46

while46:                                          ; preds = %while_body47, %while_body45
  %j56 = load i32, i32* %j
  %cols57 = load i32, i32* @cols
  %tmp58 = icmp slt i32 %j56, %cols57
  br i1 %tmp58, label %while_body47, label %merge59

while_body47:                                     ; preds = %while46
  %i48 = load i32, i32* %i
  %j49 = load i32, i32* %j
  %_board50 = getelementptr [20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9 0, i32 %i48,
i32 %j49
  store %_tile* %_board50, %_tile** %e
  %e51 = load %_tile*, %_tile** %e
  %_tile_fld_ptr52 = getelementptr inbounds %_tile, %_tile* %e51, i32 0, i32 2
  %_tcode53 = load i32, i32* %_tile_fld_ptr52
  %_fn_ptr_ptr = getelementptr [5 x i1 (%_tile*)*], [5 x i1 (%_tile*)*]* @_does_tbl, i8 0, i32
%_tcode53
  %_fn_ptr = load i1 (%_tile*)*, i1 (%_tile*)** %_fn_ptr_ptr
  %_does = call i1 %_fn_ptr(%_tile* %e51)
  %j54 = load i32, i32* %j
  %tmp55 = add i32 %j54, 1
  store i32 %tmp55, i32* %j
  br label %while46

merge59:                                          ; preds = %while46
  %i60 = load i32, i32* %i
  %tmp61 = add i32 %i60, 1
  store i32 %tmp61, i32* %i
  br label %while44
```

```
merge65:                                          ; preds = %while44
  br label %while14

merge67:                                          ; preds = %while14
  ret i32 0
}

define i1 @_nothing(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  ret i1 false
}

define i1 @_init_world() {
entry:
  %j = alloca i32
  %i = alloca i32
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %merge, %entry
  %i12 = load i32, i32* %i
  %rows13 = load i32, i32* @rows
  %tmp14 = icmp slt i32 %i12, %rows13
  br i1 %tmp14, label %while_body, label %merge15

while_body:                                       ; preds = %while
  store i32 0, i32* %j
  br label %while1

while1:                                           ; preds = %while_body2, %while_body
  %j7 = load i32, i32* %j
  %cols = load i32, i32* @cols
  %tmp8 = icmp slt i32 %j7, %cols
  br i1 %tmp8, label %while_body2, label %merge

while_body2:                                      ; preds = %while1
  %i3 = load i32, i32* %i
  %j4 = load i32, i32* %j
  %_board = getelementptr [20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9 0, i32 %i3, i32
%j4
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %_board, i32 0, i32 2
```

```
  store i32 1, i32* %_tile_fld_ptr
  %malloccall = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32))
  %_ent_ptr = bitcast i8* %malloccall to %wall*
  %_v_ent_ptr = bitcast %wall* %_ent_ptr to i8*
  %_tile_fld_ptr5 = getelementptr inbounds %_tile, %_tile* %_board, i32 0, i32 3
  store i8* %_v_ent_ptr, i8** %_tile_fld_ptr5
  %_build = call i1 @_build_wall(%_tile* %_board)
  %j6 = load i32, i32* %j
  %tmp = add i32 %j6, 1
  store i32 %tmp, i32* %j
  br label %while1

merge:                                            ; preds = %while1
  %i9 = load i32, i32* %i
  %rows = load i32, i32* @rows
  %tmp10 = sub i32 %rows, 1
  %tmp11 = add i32 %i9, %tmp10
  store i32 %tmp11, i32* %i
  br label %while

merge15:                                          ; preds = %while
  store i32 0, i32* %i
  br label %while16

while16:                                          ; preds = %merge36, %merge15
  %i39 = load i32, i32* %i
  %rows40 = load i32, i32* @rows
  %tmp41 = icmp slt i32 %i39, %rows40
  br i1 %tmp41, label %while_body17, label %merge42

while_body17:                                     ; preds = %while16
  store i32 0, i32* %j
  br label %while18

while18:                                          ; preds = %while_body19, %while_body17
  %j33 = load i32, i32* %j
  %cols34 = load i32, i32* @cols
  %tmp35 = icmp slt i32 %j33, %cols34
  br i1 %tmp35, label %while_body19, label %merge36

while_body19:                                     ; preds = %while18
  %i20 = load i32, i32* %i
  %j21 = load i32, i32* %j
```

```
  %_board22 = getelementptr [20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9 0, i32 %i20,
i32 %j21
  %_tile_fld_ptr23 = getelementptr inbounds %_tile, %_tile* %_board22, i32 0, i32 2
  store i32 1, i32* %_tile_fld_ptr23
  %malloccall24 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32))
  %_ent_ptr25 = bitcast i8* %malloccall24 to %wall*
  %_v_ent_ptr26 = bitcast %wall* %_ent_ptr25 to i8*
  %_tile_fld_ptr27 = getelementptr inbounds %_tile, %_tile* %_board22, i32 0, i32 3
  store i8* %_v_ent_ptr26, i8** %_tile_fld_ptr27
  %_build28 = call i1 @_build_wall(%_tile* %_board22)
  %j29 = load i32, i32* %j
  %cols30 = load i32, i32* @cols
  %tmp31 = sub i32 %cols30, 1
  %tmp32 = add i32 %j29, %tmp31
  store i32 %tmp32, i32* %j
  br label %while18

merge36:                                          ; preds = %while18
  %i37 = load i32, i32* %i
  %tmp38 = add i32 %i37, 1
  store i32 %tmp38, i32* %i
  br label %while16

merge42:                                          ; preds = %while16
  store i32 3, i32* getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9
0, i32 5, i32 10, i32 2)
  %malloccall43 = tail call i8* @malloc(i32 ptrtoint (%goblin* getelementptr (%goblin, %goblin*
null, i32 1) to i32))
  %_ent_ptr44 = bitcast i8* %malloccall43 to %goblin*
  %_v_ent_ptr45 = bitcast %goblin* %_ent_ptr44 to i8*
  store i8* %_v_ent_ptr45, i8** getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]*
@_board, i9 0, i32 5, i32 10, i32 3)
  %_build46 = call i1 @_build_goblin(%_tile* getelementptr inbounds ([20 x [60 x %_tile]], [20 x
[60 x %_tile]]* @_board, i9 0, i32 5, i32 10))
  store i32 3, i32* getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9
0, i32 15, i32 35, i32 2)
  %malloccall47 = tail call i8* @malloc(i32 ptrtoint (%goblin* getelementptr (%goblin, %goblin*
null, i32 1) to i32))
  %_ent_ptr48 = bitcast i8* %malloccall47 to %goblin*
  %_v_ent_ptr49 = bitcast %goblin* %_ent_ptr48 to i8*
  store i8* %_v_ent_ptr49, i8** getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]*
@_board, i9 0, i32 15, i32 35, i32 3)
```

```
  %_build50 = call i1 @_build_goblin(%_tile* getelementptr inbounds ([20 x [60 x %_tile]], [20 x
[60 x %_tile]]* @_board, i9 0, i32 15, i32 35))
  store i32 3, i32* getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9
0, i32 16, i32 15, i32 2)
  %malloccall51 = tail call i8* @malloc(i32 ptrtoint (%goblin* getelementptr (%goblin, %goblin*
null, i32 1) to i32))
  %_ent_ptr52 = bitcast i8* %malloccall51 to %goblin*
  %_v_ent_ptr53 = bitcast %goblin* %_ent_ptr52 to i8*
  store i8* %_v_ent_ptr53, i8** getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]*
@_board, i9 0, i32 16, i32 15, i32 3)
  %_build54 = call i1 @_build_goblin(%_tile* getelementptr inbounds ([20 x [60 x %_tile]], [20 x
[60 x %_tile]]* @_board, i9 0, i32 16, i32 15))
  store i32 2, i32* getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9
0, i32 11, i32 35, i32 2)
  %malloccall55 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32))
  %_ent_ptr56 = bitcast i8* %malloccall55 to %trap*
  %_v_ent_ptr57 = bitcast %trap* %_ent_ptr56 to i8*
  store i8* %_v_ent_ptr57, i8** getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]*
@_board, i9 0, i32 11, i32 35, i32 3)
  %_build58 = call i1 @_build_trap(%_tile* getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60
x %_tile]]* @_board, i9 0, i32 11, i32 35))
  store i32 2, i32* getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9
0, i32 15, i32 20, i32 2)
  %malloccall59 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32))
  %_ent_ptr60 = bitcast i8* %malloccall59 to %trap*
  %_v_ent_ptr61 = bitcast %trap* %_ent_ptr60 to i8*
  store i8* %_v_ent_ptr61, i8** getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]*
@_board, i9 0, i32 15, i32 20, i32 3)
  %_build62 = call i1 @_build_trap(%_tile* getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60
x %_tile]]* @_board, i9 0, i32 15, i32 20))
  store i32 2, i32* getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9
0, i32 5, i32 5, i32 2)
  %malloccall63 = tail call i8* @malloc(i32 ptrtoint (i1* getelementptr (i1, i1* null, i32 1) to i32))
  %_ent_ptr64 = bitcast i8* %malloccall63 to %trap*
  %_v_ent_ptr65 = bitcast %trap* %_ent_ptr64 to i8*
  store i8* %_v_ent_ptr65, i8** getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]*
@_board, i9 0, i32 5, i32 5, i32 3)
  %_build66 = call i1 @_build_trap(%_tile* getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60
x %_tile]]* @_board, i9 0, i32 5, i32 5))
  store i32 4, i32* getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9
0, i32 5, i32 5, i32 2)
  %malloccall67 = tail call i8* @malloc(i32 ptrtoint (%player* getelementptr (%player, %player*
null, i32 1) to i32))
```

```
  %_ent_ptr68 = bitcast i8* %malloccall67 to %player*
  %_v_ent_ptr69 = bitcast %player* %_ent_ptr68 to i8*
  store i8* %_v_ent_ptr69, i8** getelementptr inbounds ([20 x [60 x %_tile]], [20 x [60 x %_tile]]*
@_board, i9 0, i32 5, i32 5, i32 3)
  %_build70 = call i1 @_build_player(%_tile* getelementptr inbounds ([20 x [60 x %_tile]], [20 x
[60 x %_tile]]* @_board, i9 0, i32 5, i32 5))
  ret i1 false
}

define i1 @_does_player(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  %this2 = load %_tile*, %_tile** %this1
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %this2, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr
  %_ent_ptr = bitcast i8* %_void_ptr to i1*
  %_called = load i1, i1* %_ent_ptr
  %tmp = xor i1 %_called, true
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %then
  ret i1 false

then:                                             ; preds = %entry
  %this3 = load %_tile*, %_tile** %this1
  %_tile_fld_ptr4 = getelementptr inbounds %_tile, %_tile* %this3, i32 0, i32 3
  %_void_ptr5 = load i8*, i8** %_tile_fld_ptr4
  %_ent_ptr6 = bitcast i8* %_void_ptr5 to i1*
  store i1 true, i1* %_ent_ptr6
  %this7 = load %_tile*, %_tile** %this1
  %playerMove_result = call i1 @playerMove(%_tile* %this7)
  br label %merge

else:                                             ; preds = %entry
  br label %merge
}

define i1 @_does_goblin(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  %this2 = load %_tile*, %_tile** %this1
```

```
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %this2, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr
  %_ent_ptr = bitcast i8* %_void_ptr to i1*
  %_called = load i1, i1* %_ent_ptr
  %tmp = xor i1 %_called, true
  br i1 %tmp, label %then, label %else

merge:                                            ; preds = %else, %then
  ret i1 false

then:                                             ; preds = %entry
  %this3 = load %_tile*, %_tile** %this1
  %_tile_fld_ptr4 = getelementptr inbounds %_tile, %_tile* %this3, i32 0, i32 3
  %_void_ptr5 = load i8*, i8** %_tile_fld_ptr4
  %_ent_ptr6 = bitcast i8* %_void_ptr5 to i1*
  store i1 true, i1* %_ent_ptr6
  %this7 = load %_tile*, %_tile** %this1
  %goblinMove_result = call i1 @goblinMove(%_tile* %this7)
  br label %merge

else:                                             ; preds = %entry
  br label %merge
}

define i1 @_does_trap(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  ret i1 false
}

define i1 @_does_wall(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  ret i1 false
}

define i1 @_build_player(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  %this2 = load %_tile*, %_tile** %this1
```

```
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %this2, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr
  %_ent_ptr = bitcast i8* %_void_ptr to %player*
  %_ent_fld_ptr = getelementptr inbounds %player, %player* %_ent_ptr, i32 0, i32 1
  store i32 5, i32* %_ent_fld_ptr
  %this3 = load %_tile*, %_tile** %this1
  %_tile_fld_ptr4 = getelementptr inbounds %_tile, %_tile* %this3, i32 0, i32 3
  %_void_ptr5 = load i8*, i8** %_tile_fld_ptr4
  %_ent_ptr6 = bitcast i8* %_void_ptr5 to %player*
  %_ent_fld_ptr7 = getelementptr inbounds %player, %player* %_ent_ptr6, i32 0, i32 2
  store i32 3, i32* %_ent_fld_ptr7
  ret i1 false
}

define i1 @_build_goblin(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  %this2 = load %_tile*, %_tile** %this1
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %this2, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr
  %_ent_ptr = bitcast i8* %_void_ptr to %goblin*
  %_ent_fld_ptr = getelementptr inbounds %goblin, %goblin* %_ent_ptr, i32 0, i32 1
  store i32 5, i32* %_ent_fld_ptr
  %this3 = load %_tile*, %_tile** %this1
  %_tile_fld_ptr4 = getelementptr inbounds %_tile, %_tile* %this3, i32 0, i32 3
  %_void_ptr5 = load i8*, i8** %_tile_fld_ptr4
  %_ent_ptr6 = bitcast i8* %_void_ptr5 to %goblin*
  %_ent_fld_ptr7 = getelementptr inbounds %goblin, %goblin* %_ent_ptr6, i32 0, i32 2
  store i32 0, i32* %_ent_fld_ptr7
  ret i1 false
}

define i1 @_build_trap(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  ret i1 false
}

define i1 @_build_wall(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
```

```
  store %_tile* %this, %_tile** %this1
  ret i1 false
}

define i1 @moveOrAttack(%_tile* %p, i32 %r, i32 %c) {
entry:
  %p1 = alloca %_tile*
  store %_tile* %p, %_tile** %p1
  %r2 = alloca i32
  store i32 %r, i32* %r2
  %c3 = alloca i32
  store i32 %c, i32* %c3
  %e = alloca %_tile*
  %r4 = load i32, i32* %r2
  %c5 = load i32, i32* %c3
  %_board = getelementptr [20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9 0, i32 %r4, i32
%c5
  store %_tile* %_board, %_tile** %e
  %e6 = load %_tile*, %_tile** %e
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %e6, i32 0, i32 2
  %_e_type_ptr = load i32, i32* %_tile_fld_ptr
  %_tmp = icmp eq i32 %_e_type_ptr, 3
  br i1 %_tmp, label %then, label %else41

merge:                                            ; preds = %merge46, %merge26
  ret i1 true

then:                                             ; preds = %entry
  %e7 = load %_tile*, %_tile** %e
  %e8 = load %_tile*, %_tile** %e
  %_tile_fld_ptr9 = getelementptr inbounds %_tile, %_tile* %e8, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr9
  %_ent_ptr = bitcast i8* %_void_ptr to %goblin*
  %_ent_fld_ptr = getelementptr inbounds %goblin, %goblin* %_ent_ptr, i32 0, i32 1
  %health = load i32, i32* %_ent_fld_ptr
  %p10 = load %_tile*, %_tile** %p1
  %_tile_fld_ptr11 = getelementptr inbounds %_tile, %_tile* %p10, i32 0, i32 3
  %_void_ptr12 = load i8*, i8** %_tile_fld_ptr11
  %_ent_ptr13 = bitcast i8* %_void_ptr12 to %player*
  %_ent_fld_ptr14 = getelementptr inbounds %player, %player* %_ent_ptr13, i32 0, i32 1
  %attack = load i32, i32* %_ent_fld_ptr14
  %tmp = sub i32 %health, %attack
  %_tile_fld_ptr15 = getelementptr inbounds %_tile, %_tile* %e7, i32 0, i32 3
```

```
  %_void_ptr16 = load i8*, i8** %_tile_fld_ptr15
  %_ent_ptr17 = bitcast i8* %_void_ptr16 to %goblin*
  %_ent_fld_ptr18 = getelementptr inbounds %goblin, %goblin* %_ent_ptr17, i32 0, i32 1
  store i32 %tmp, i32* %_ent_fld_ptr18
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([26 x i8], [26 x i8]* @.str.24, i32
0, i32 0))
  %e19 = load %_tile*, %_tile** %e
  %_tile_fld_ptr20 = getelementptr inbounds %_tile, %_tile* %e19, i32 0, i32 3
  %_void_ptr21 = load i8*, i8** %_tile_fld_ptr20
  %_ent_ptr22 = bitcast i8* %_void_ptr21 to %goblin*
  %_ent_fld_ptr23 = getelementptr inbounds %goblin, %goblin* %_ent_ptr22, i32 0, i32 1
  %health24 = load i32, i32* %_ent_fld_ptr23
  %tmp25 = icmp sle i32 %health24, 0
  br i1 %tmp25, label %then27, label %else

merge26:                                          ; preds = %else, %then27
  br label %merge

then27:                                           ; preds = %then
  %printf28 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([42 x i8], [42 x i8]* @.str.25,
i32 0, i32 0))
  %getKey = call i8 @getKey()
  %e29 = load %_tile*, %_tile** %e
  %_tile_fld_ptr30 = getelementptr inbounds %_tile, %_tile* %e29, i32 0, i32 2
  %_tile_fld_ptr31 = getelementptr inbounds %_tile, %_tile* %e29, i32 0, i32 3
  store i32 0, i32* %_tile_fld_ptr30
  store i8* null, i8** %_tile_fld_ptr31
  %_ptr = load i8*, i8** %_tile_fld_ptr31
  tail call void @free(i8* %_ptr)
  %p32 = load %_tile*, %_tile** %p1
  %r33 = load i32, i32* %r2
  %c34 = load i32, i32* %c3
  %_board35 = getelementptr [20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9 0, i32 %r33,
i32 %c34
  %_tile_fld_ptr36 = getelementptr inbounds %_tile, %_tile* %p32, i32 0, i32 2
  %_tile_fld_ptr37 = getelementptr inbounds %_tile, %_tile* %p32, i32 0, i32 3
  %_typ = load i32, i32* %_tile_fld_ptr36
  %_ptr38 = load i8*, i8** %_tile_fld_ptr37
  %_tile_fld_ptr39 = getelementptr inbounds %_tile, %_tile* %_board35, i32 0, i32 2
  %_tile_fld_ptr40 = getelementptr inbounds %_tile, %_tile* %_board35, i32 0, i32 3
  store i32 %_typ, i32* %_tile_fld_ptr39
  store i8* %_ptr38, i8** %_tile_fld_ptr40
  store i32 0, i32* %_tile_fld_ptr36
```

```
  store i8* null, i8** %_tile_fld_ptr37
  br label %merge26

else:                                             ; preds = %then
  br label %merge26

else41:                                           ; preds = %entry
  %e42 = load %_tile*, %_tile** %e
  %_tile_fld_ptr43 = getelementptr inbounds %_tile, %_tile* %e42, i32 0, i32 2
  %_e_type_ptr44 = load i32, i32* %_tile_fld_ptr43
  %_tmp45 = icmp eq i32 %_e_type_ptr44, 2
  br i1 %_tmp45, label %then47, label %else49

merge46:                                          ; preds = %merge54, %then47
  br label %merge

then47:                                           ; preds = %else41
  %printf48 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([34 x i8], [34 x i8]* @.str.26,
i32 0, i32 0))
  store i1 true, i1* @_exit
  br label %merge46

else49:                                           ; preds = %else41
  %e50 = load %_tile*, %_tile** %e
  %_tile_fld_ptr51 = getelementptr inbounds %_tile, %_tile* %e50, i32 0, i32 2
  %_e_type_ptr52 = load i32, i32* %_tile_fld_ptr51
  %_tmp53 = icmp eq i32 %_e_type_ptr52, 0
  br i1 %_tmp53, label %then55, label %else66

merge54:                                          ; preds = %then55
  br label %merge46

then55:                                           ; preds = %else49
  %p56 = load %_tile*, %_tile** %p1
  %r57 = load i32, i32* %r2
  %c58 = load i32, i32* %c3
  %_board59 = getelementptr [20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9 0, i32 %r57,
i32 %c58
  %_tile_fld_ptr60 = getelementptr inbounds %_tile, %_tile* %p56, i32 0, i32 2
  %_tile_fld_ptr61 = getelementptr inbounds %_tile, %_tile* %p56, i32 0, i32 3
  %_typ62 = load i32, i32* %_tile_fld_ptr60
  %_ptr63 = load i8*, i8** %_tile_fld_ptr61
  %_tile_fld_ptr64 = getelementptr inbounds %_tile, %_tile* %_board59, i32 0, i32 2
```

```
  %_tile_fld_ptr65 = getelementptr inbounds %_tile, %_tile* %_board59, i32 0, i32 3
  store i32 %_typ62, i32* %_tile_fld_ptr64
  store i8* %_ptr63, i8** %_tile_fld_ptr65
  store i32 0, i32* %_tile_fld_ptr60
  store i8* null, i8** %_tile_fld_ptr61
  br label %merge54

else66:                                           ; preds = %else49
  ret i1 false
}

define i1 @moveIfEmpty(%_tile* %p, i32 %r, i32 %c) {
entry:
  %p1 = alloca %_tile*
  store %_tile* %p, %_tile** %p1
  %r2 = alloca i32
  store i32 %r, i32* %r2
  %c3 = alloca i32
  store i32 %c, i32* %c3
  %e = alloca %_tile*
  %r4 = load i32, i32* %r2
  %c5 = load i32, i32* %c3
  %_board = getelementptr [20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9 0, i32 %r4, i32
%c5
  store %_tile* %_board, %_tile** %e
  %e6 = load %_tile*, %_tile** %e
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %e6, i32 0, i32 2
  %_e_type_ptr = load i32, i32* %_tile_fld_ptr
  %_tmp = icmp eq i32 %_e_type_ptr, 0
  br i1 %_tmp, label %then, label %else

merge:                                            ; preds = %else
  ret i1 false

then:                                             ; preds = %entry
  %p7 = load %_tile*, %_tile** %p1
  %r8 = load i32, i32* %r2
  %c9 = load i32, i32* %c3
  %_board10 = getelementptr [20 x [60 x %_tile]], [20 x [60 x %_tile]]* @_board, i9 0, i32 %r8,
i32 %c9
  %_tile_fld_ptr11 = getelementptr inbounds %_tile, %_tile* %p7, i32 0, i32 2
  %_tile_fld_ptr12 = getelementptr inbounds %_tile, %_tile* %p7, i32 0, i32 3
  %_typ = load i32, i32* %_tile_fld_ptr11
```

```
  %_ptr = load i8*, i8** %_tile_fld_ptr12
  %_tile_fld_ptr13 = getelementptr inbounds %_tile, %_tile* %_board10, i32 0, i32 2
  %_tile_fld_ptr14 = getelementptr inbounds %_tile, %_tile* %_board10, i32 0, i32 3
  store i32 %_typ, i32* %_tile_fld_ptr13
  store i8* %_ptr, i8** %_tile_fld_ptr14
  store i32 0, i32* %_tile_fld_ptr11
  store i8* null, i8** %_tile_fld_ptr12
  ret i1 true

else:                                             ; preds = %entry
  br label %merge
}

define i1 @goblinMove(%_tile* %p) {
entry:
  %p1 = alloca %_tile*
  store %_tile* %p, %_tile** %p1
  %r = alloca i32
  %c = alloca i32
  %k = alloca i32
  %p2 = load %_tile*, %_tile** %p1
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %p2, i32 0, i32 0
  %_row = load i32, i32* %_tile_fld_ptr
  store i32 %_row, i32* %r
  %p3 = load %_tile*, %_tile** %p1
  %_tile_fld_ptr4 = getelementptr inbounds %_tile, %_tile* %p3, i32 0, i32 1
  %_col = load i32, i32* %_tile_fld_ptr4
  store i32 %_col, i32* %c
  %r5 = load i32, i32* %r
  %tmp = srem i32 %r5, 2
  %tmp6 = icmp eq i32 %tmp, 1
  br i1 %tmp6, label %then, label %else

merge:                                            ; preds = %else, %then
  %k7 = load i32, i32* %k
  %tmp8 = icmp eq i32 %k7, 0
  br i1 %tmp8, label %then10, label %else15

then:                                             ; preds = %entry
  store i32 1, i32* %k
  br label %merge

else:                                             ; preds = %entry
```

```
  br label %merge

merge9:                                           ; preds = %merge18, %then10
  ret i1 true

then10:                                           ; preds = %merge
  %c11 = load i32, i32* %c
  %r12 = load i32, i32* %r
  %tmp13 = sub i32 %r12, 1
  %p14 = load %_tile*, %_tile** %p1
  %moveIfEmpty_result = call i1 @moveIfEmpty(%_tile* %p14, i32 %tmp13, i32 %c11)
  store i32 1, i32* %k
  br label %merge9

else15:                                           ; preds = %merge
  %k16 = load i32, i32* %k
  %tmp17 = icmp eq i32 %k16, 1
  br i1 %tmp17, label %then19, label %else25

merge18:                                          ; preds = %merge28, %then19
  br label %merge9

then19:                                           ; preds = %else15
  %c20 = load i32, i32* %c
  %tmp21 = sub i32 %c20, 1
  %r22 = load i32, i32* %r
  %p23 = load %_tile*, %_tile** %p1
  %moveIfEmpty_result24 = call i1 @moveIfEmpty(%_tile* %p23, i32 %r22, i32 %tmp21)
  store i32 2, i32* %k
  br label %merge18

else25:                                           ; preds = %else15
  %k26 = load i32, i32* %k
  %tmp27 = icmp eq i32 %k26, 2
  br i1 %tmp27, label %then29, label %else35

merge28:                                          ; preds = %merge38, %then29
  br label %merge18

then29:                                           ; preds = %else25
  %c30 = load i32, i32* %c
  %r31 = load i32, i32* %r
  %tmp32 = add i32 %r31, 1
```

```
  %p33 = load %_tile*, %_tile** %p1
  %moveIfEmpty_result34 = call i1 @moveIfEmpty(%_tile* %p33, i32 %tmp32, i32 %c30)
  store i32 3, i32* %k
  br label %merge28

else35:                                           ; preds = %else25
  %k36 = load i32, i32* %k
  %tmp37 = icmp eq i32 %k36, 3
  br i1 %tmp37, label %then39, label %else45

merge38:                                          ; preds = %merge48, %then39
  br label %merge28

then39:                                           ; preds = %else35
  %c40 = load i32, i32* %c
  %tmp41 = add i32 %c40, 1
  %r42 = load i32, i32* %r
  %p43 = load %_tile*, %_tile** %p1
  %moveIfEmpty_result44 = call i1 @moveIfEmpty(%_tile* %p43, i32 %r42, i32 %tmp41)
  store i32 4, i32* %k
  br label %merge38

else45:                                           ; preds = %else35
  %k46 = load i32, i32* %k
  %tmp47 = icmp eq i32 %k46, 4
  br i1 %tmp47, label %then49, label %else50

merge48:                                          ; preds = %else50, %then49
  br label %merge38

then49:                                           ; preds = %else45
  store i32 0, i32* %k
  br label %merge48

else50:                                           ; preds = %else45
  br label %merge48
}

define i1 @playerMove(%_tile* %p) {
entry:
  %p1 = alloca %_tile*
  store %_tile* %p, %_tile** %p1
  %r = alloca i32
```

```
  %c = alloca i32
  %k = alloca i8
  %moved = alloca i1
  store i1 false, i1* %moved
  %p2 = load %_tile*, %_tile** %p1
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %p2, i32 0, i32 0
  %_row = load i32, i32* %_tile_fld_ptr
  store i32 %_row, i32* %r
  %p3 = load %_tile*, %_tile** %p1
  %_tile_fld_ptr4 = getelementptr inbounds %_tile, %_tile* %p3, i32 0, i32 1
  %_col = load i32, i32* %_tile_fld_ptr4
  store i32 %_col, i32* %c
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([33 x i8], [33 x i8]* @.str.33, i32
0, i32 0))
  br label %while

while:                                            ; preds = %merge, %entry
  %moved71 = load i1, i1* %moved
  %tmp72 = xor i1 %moved71, true
  br i1 %tmp72, label %while_body, label %merge73

while_body:                                       ; preds = %while
  %getKey = call i8 @getKey()
  store i8 %getKey, i8* %k
  %k5 = load i8, i8* %k
  %tmp = icmp eq i8 %k5, 119
  %r6 = load i32, i32* %r
  %tmp7 = icmp sgt i32 %r6, 0
  %tmp8 = and i1 %tmp, %tmp7
  br i1 %tmp8, label %then, label %else15

merge:                                            ; preds = %merge21, %merge13
  br label %while

then:                                             ; preds = %while_body
  %c9 = load i32, i32* %c
  %r10 = load i32, i32* %r
  %tmp11 = sub i32 %r10, 1
  %p12 = load %_tile*, %_tile** %p1
  %moveOrAttack_result = call i1 @moveOrAttack(%_tile* %p12, i32 %tmp11, i32 %c9)
  br i1 %moveOrAttack_result, label %then14, label %else

merge13:                                          ; preds = %else, %then14
```

```
  br label %merge

then14:                                           ; preds = %then
  store i1 true, i1* %moved
  br label %merge13

else:                                             ; preds = %then
  br label %merge13

else15:                                           ; preds = %while_body
  %k16 = load i8, i8* %k
  %tmp17 = icmp eq i8 %k16, 97
  %c18 = load i32, i32* %c
  %tmp19 = icmp sgt i32 %c18, 0
  %tmp20 = and i1 %tmp17, %tmp19
  br i1 %tmp20, label %then22, label %else31

merge21:                                          ; preds = %merge38, %merge28
  br label %merge

then22:                                           ; preds = %else15
  %c23 = load i32, i32* %c
  %tmp24 = sub i32 %c23, 1
  %r25 = load i32, i32* %r
  %p26 = load %_tile*, %_tile** %p1
  %moveOrAttack_result27 = call i1 @moveOrAttack(%_tile* %p26, i32 %r25, i32 %tmp24)
  br i1 %moveOrAttack_result27, label %then29, label %else30

merge28:                                          ; preds = %else30, %then29
  br label %merge21

then29:                                           ; preds = %then22
  store i1 true, i1* %moved
  br label %merge28

else30:                                           ; preds = %then22
  br label %merge28

else31:                                           ; preds = %else15
  %k32 = load i8, i8* %k
  %tmp33 = icmp eq i8 %k32, 115
  %r34 = load i32, i32* %r
  %rows = load i32, i32* @rows
```

```
  %tmp35 = sub i32 %rows, 1
  %tmp36 = icmp slt i32 %r34, %tmp35
  %tmp37 = and i1 %tmp33, %tmp36
  br i1 %tmp37, label %then39, label %else48

merge38:                                          ; preds = %merge55, %merge45
  br label %merge21

then39:                                           ; preds = %else31
  %c40 = load i32, i32* %c
  %r41 = load i32, i32* %r
  %tmp42 = add i32 %r41, 1
  %p43 = load %_tile*, %_tile** %p1
  %moveOrAttack_result44 = call i1 @moveOrAttack(%_tile* %p43, i32 %tmp42, i32 %c40)
  br i1 %moveOrAttack_result44, label %then46, label %else47

merge45:                                          ; preds = %else47, %then46
  br label %merge38

then46:                                           ; preds = %then39
  store i1 true, i1* %moved
  br label %merge45

else47:                                           ; preds = %then39
  br label %merge45

else48:                                           ; preds = %else31
  %k49 = load i8, i8* %k
  %tmp50 = icmp eq i8 %k49, 100
  %r51 = load i32, i32* %r
  %cols = load i32, i32* @cols
  %tmp52 = sub i32 %cols, 1
  %tmp53 = icmp slt i32 %r51, %tmp52
  %tmp54 = and i1 %tmp50, %tmp53
  br i1 %tmp54, label %then56, label %else65

merge55:                                          ; preds = %merge68, %merge62
  br label %merge38

then56:                                           ; preds = %else48
  %c57 = load i32, i32* %c
  %tmp58 = add i32 %c57, 1
  %r59 = load i32, i32* %r
```

```
  %p60 = load %_tile*, %_tile** %p1
  %moveOrAttack_result61 = call i1 @moveOrAttack(%_tile* %p60, i32 %r59, i32 %tmp58)
  br i1 %moveOrAttack_result61, label %then63, label %else64

merge62:                                          ; preds = %else64, %then63
  br label %merge55

then63:                                           ; preds = %then56
  store i1 true, i1* %moved
  br label %merge62

else64:                                           ; preds = %then56
  br label %merge62

else65:                                           ; preds = %else48
  %k66 = load i8, i8* %k
  %tmp67 = icmp eq i8 %k66, 113
  br i1 %tmp67, label %then69, label %else70

merge68:                                          ; preds = %else70, %then69
  br label %merge55

then69:                                           ; preds = %else65
  store i1 true, i1* @_exit
  store i1 true, i1* %moved
  br label %merge68

else70:                                           ; preds = %else65
  br label %merge68

merge73:                                          ; preds = %while
  ret i1 true
}

declare noalias i8* @malloc(i32)

declare void @free(i8*)
```

Source Program 2: Goblin Eater

```
world[20,40] {
    num i;
```

```
    num j;
    for(i = 0; i < 20; i += 1) {
      for(j = 0; j < 40; j += 1) {
        place(goblin, i, j);
      }
    }
    place(player, 10, 30);
}
entities {
  @:player {
    num attack;
    build {
      this.attack = 5;
    }
    does {
      playerMove(this);
      prints("Press q to quit or any other key to end turn");
      if(getKey() == 'q') {
        exit;
      }
    }
  }
  g:goblin {
    num health;
    build {
      this.health = 10;
    }
  }
}
functions {
  bool playerMove(entity p) {
    num r;
    num c;
    char k;
    bool moved;
    moved = false;
    r = row(p);
    c = col(p);
    prints("Press wasd to move or q to quit");
    while(not moved) {
      k = getKey();
      if(k == 'w' and r > 0) {
        moveOrAttack(p, r - 1, c);
```

```
      moved = true;
    } else if(k == 'a' and c > 0) {
      moveOrAttack(p, r, c - 1);
      moved = true;
    } else if(k == 's' and r < rows) {
      moveOrAttack(p, r + 1, c);
      moved = true;
    } else if(k == 'd' and c < cols) {
      moveOrAttack(p, r, c + 1);
      moved = true;
    } else if(k == 'q') {
      exit;
      moved = true;
    }
  }
  return true;
 }
 bool moveOrAttack(entity p, num r, num c) {
  entity e;
  e = peek(r, c);
  if(e is empty) {
    move(p, r, c);
  } else if(e is goblin) {
    e.health -= p.attack;
    prints("dealt 5 damage to goblin");
    if(e.health <= 0) {
      prints("killed goblin");
      remove(e);
      move(p, r, c);
    }
  }
 }
}
```

LLVM Program 2: Goblin Eater

```
; ModuleID = 'Goblin'

%_tile = type { i32, i32, i32, i8* }
%goblin = type { i1, i32 }
%player = type { i1, i32 }

@_board = global [20 x [40 x %_tile]] zeroinitializer
```

```
@rows = global i32 0
@cols = global i32 0
@_exit = global i1 false
@_symbols = global [3 x i8] c" g@"
@_build_tbl = global [3 x i1 (%_tile*)*] [i1 (%_tile*)* @_nothing, i1 (%_tile*)* @_build_goblin, i1
(%_tile*)* @_build_player]
@_does_tbl = global [3 x i1 (%_tile*)*] [i1 (%_tile*)* @_nothing, i1 (%_tile*)* @_does_goblin, i1
(%_tile*)* @_does_player]
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [3 x i8] c"%c\00"
@.str = private unnamed_addr constant [2 x i8] c"\0A\00"
@fmt.2 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.3 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.4 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.5 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.6 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.7 = private unnamed_addr constant [3 x i8] c"%c\00"
@.str.8 = private unnamed_addr constant [46 x i8] c"Press q to quit or any other key to end
turn\0A\00"
@fmt.9 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.10 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.11 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.12 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.13 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.14 = private unnamed_addr constant [3 x i8] c"%c\00"
@fmt.15 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.16 = private unnamed_addr constant [3 x i8] c"%c\00"
@.str.17 = private unnamed_addr constant [26 x i8] c"dealt 5 damage to goblin\0A\00"
@.str.18 = private unnamed_addr constant [15 x i8] c"killed goblin\0A\00"
@fmt.19 = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.20 = private unnamed_addr constant [3 x i8] c"%c\00"
@.str.21 = private unnamed_addr constant [33 x i8] c"Press wasd to move or q to quit\0A\00"

declare i32 @printf(i8*, ...)

declare i8 @getKey()

declare i32 @_clearScreen()

define i32 @main() {
entry:
  %i = alloca i32
  %j = alloca i32
```

```
  %e = alloca %_tile*
  store i32 20, i32* @rows
  store i32 40, i32* @cols
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %merge, %entry
  %i11 = load i32, i32* %i
  %rows = load i32, i32* @rows
  %tmp12 = icmp slt i32 %i11, %rows
  br i1 %tmp12, label %while_body, label %merge13

while_body:                                       ; preds = %while
  store i32 0, i32* %j
  br label %while1

while1:                                           ; preds = %while_body2, %while_body
  %j7 = load i32, i32* %j
  %cols = load i32, i32* @cols
  %tmp8 = icmp slt i32 %j7, %cols
  br i1 %tmp8, label %while_body2, label %merge

while_body2:                                      ; preds = %while1
  %i3 = load i32, i32* %i
  %j4 = load i32, i32* %j
  %_board = getelementptr [20 x [40 x %_tile]], [20 x [40 x %_tile]]* @_board, i1 false, i32 %i3,
i32 %j4
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %_board, i32 0, i32 0
  store i32 %i3, i32* %_tile_fld_ptr
  %_tile_fld_ptr5 = getelementptr inbounds %_tile, %_tile* %_board, i32 0, i32 1
  store i32 %j4, i32* %_tile_fld_ptr5
  %j6 = load i32, i32* %j
  %tmp = add i32 %j6, 1
  store i32 %tmp, i32* %j
  br label %while1

merge:                                            ; preds = %while1
  %i9 = load i32, i32* %i
  %tmp10 = add i32 %i9, 1
  store i32 %tmp10, i32* %i
  br label %while

merge13:                                          ; preds = %while
```

```
  %_init_world_result = call i1 @_init_world()
  br label %while14

while14:                                          ; preds = %merge65, %merge13
  %_exit = load i1, i1* @_exit
  %tmp66 = xor i1 %_exit, true
  br i1 %tmp66, label %while_body15, label %merge67

while_body15:                                     ; preds = %while14
  %_clearScreen = call i32 @_clearScreen()
  store i32 0, i32* %i
  br label %while16

while16:                                          ; preds = %merge36, %while_body15
  %i40 = load i32, i32* %i
  %rows41 = load i32, i32* @rows
  %tmp42 = icmp slt i32 %i40, %rows41
  br i1 %tmp42, label %while_body17, label %merge43

while_body17:                                     ; preds = %while16
  store i32 0, i32* %j
  br label %while18

while18:                                          ; preds = %merge26, %while_body17
  %j33 = load i32, i32* %j
  %cols34 = load i32, i32* @cols
  %tmp35 = icmp slt i32 %j33, %cols34
  br i1 %tmp35, label %while_body19, label %merge36

while_body19:                                     ; preds = %while18
  %i20 = load i32, i32* %i
  %j21 = load i32, i32* %j
  %_board22 = getelementptr [20 x [40 x %_tile]], [20 x [40 x %_tile]]* @_board, i9 0, i32 %i20,
i32 %j21
  store %_tile* %_board22, %_tile** %e
  %e23 = load %_tile*, %_tile** %e
  %_tile_fld_ptr24 = getelementptr inbounds %_tile, %_tile* %e23, i32 0, i32 2
  %_e_type_ptr = load i32, i32* %_tile_fld_ptr24
  %_tmp = icmp eq i32 %_e_type_ptr, 0
  %tmp25 = xor i1 %_tmp, true
  br i1 %tmp25, label %then, label %else

merge26:                                          ; preds = %else, %then
```

```
  %e29 = load %_tile*, %_tile** %e
  %_tile_fld_ptr30 = getelementptr inbounds %_tile, %_tile* %e29, i32 0, i32 2
  %_tcode = load i32, i32* %_tile_fld_ptr30
  %_sym_ptr = getelementptr [3 x i8], [3 x i8]* @_symbols, i8 0, i32 %_tcode
  %_sym = load i8, i8* %_sym_ptr
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt.1, i32 0,
i32 0), i8 %_sym)
  %j31 = load i32, i32* %j
  %tmp32 = add i32 %j31, 1
  store i32 %tmp32, i32* %j
  br label %while18

then:                                             ; preds = %while_body19
  %e27 = load %_tile*, %_tile** %e
  %_tile_fld_ptr28 = getelementptr inbounds %_tile, %_tile* %e27, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr28
  %_ent_ptr = bitcast i8* %_void_ptr to i1*
  store i1 false, i1* %_ent_ptr
  br label %merge26

else:                                             ; preds = %while_body19
  br label %merge26

merge36:                                          ; preds = %while18
  %printf37 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([2 x i8], [2 x i8]* @.str, i32 0,
i32 0))
  %i38 = load i32, i32* %i
  %tmp39 = add i32 %i38, 1
  store i32 %tmp39, i32* %i
  br label %while16

merge43:                                          ; preds = %while16
  store i32 0, i32* %i
  br label %while44

while44:                                          ; preds = %merge59, %merge43
  %i62 = load i32, i32* %i
  %rows63 = load i32, i32* @rows
  %tmp64 = icmp slt i32 %i62, %rows63
  br i1 %tmp64, label %while_body45, label %merge65

while_body45:                                     ; preds = %while44
  store i32 0, i32* %j
```

```
  br label %while46

while46:                                          ; preds = %while_body47, %while_body45
  %j56 = load i32, i32* %j
  %cols57 = load i32, i32* @cols
  %tmp58 = icmp slt i32 %j56, %cols57
  br i1 %tmp58, label %while_body47, label %merge59

while_body47:                                     ; preds = %while46
  %i48 = load i32, i32* %i
  %j49 = load i32, i32* %j
  %_board50 = getelementptr [20 x [40 x %_tile]], [20 x [40 x %_tile]]* @_board, i9 0, i32 %i48,
i32 %j49
  store %_tile* %_board50, %_tile** %e
  %e51 = load %_tile*, %_tile** %e
  %_tile_fld_ptr52 = getelementptr inbounds %_tile, %_tile* %e51, i32 0, i32 2
  %_tcode53 = load i32, i32* %_tile_fld_ptr52
  %_fn_ptr_ptr = getelementptr [3 x i1 (%_tile*)*], [3 x i1 (%_tile*)*]* @_does_tbl, i8 0, i32
%_tcode53
  %_fn_ptr = load i1 (%_tile*)*, i1 (%_tile*)** %_fn_ptr_ptr
  %_does = call i1 %_fn_ptr(%_tile* %e51)
  %j54 = load i32, i32* %j
  %tmp55 = add i32 %j54, 1
  store i32 %tmp55, i32* %j
  br label %while46

merge59:                                          ; preds = %while46
  %i60 = load i32, i32* %i
  %tmp61 = add i32 %i60, 1
  store i32 %tmp61, i32* %i
  br label %while44

merge65:                                          ; preds = %while44
  br label %while14

merge67:                                          ; preds = %while14
  ret i32 0
}

define i1 @_nothing(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
```

```
  ret i1 false
}

define i1 @_init_world() {
entry:
  %j = alloca i32
  %i = alloca i32
  store i32 0, i32* %i
  br label %while

while:                                            ; preds = %merge, %entry
  %i11 = load i32, i32* %i
  %tmp12 = icmp slt i32 %i11, 20
  br i1 %tmp12, label %while_body, label %merge13

while_body:                                       ; preds = %while
  store i32 0, i32* %j
  br label %while1

while1:                                           ; preds = %while_body2, %while_body
  %j7 = load i32, i32* %j
  %tmp8 = icmp slt i32 %j7, 40
  br i1 %tmp8, label %while_body2, label %merge

while_body2:                                      ; preds = %while1
  %i3 = load i32, i32* %i
  %j4 = load i32, i32* %j
  %_board = getelementptr [20 x [40 x %_tile]], [20 x [40 x %_tile]]* @_board, i9 0, i32 %i3, i32
%j4
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %_board, i32 0, i32 2
  store i32 1, i32* %_tile_fld_ptr
  %malloccall = tail call i8* @malloc(i32 ptrtoint (%goblin* getelementptr (%goblin, %goblin* null,
i32 1) to i32))
  %_ent_ptr = bitcast i8* %malloccall to %goblin*
  %_v_ent_ptr = bitcast %goblin* %_ent_ptr to i8*
  %_tile_fld_ptr5 = getelementptr inbounds %_tile, %_tile* %_board, i32 0, i32 3
  store i8* %_v_ent_ptr, i8** %_tile_fld_ptr5
  %_build = call i1 @_build_goblin(%_tile* %_board)
  %j6 = load i32, i32* %j
  %tmp = add i32 %j6, 1
  store i32 %tmp, i32* %j
  br label %while1
```

```
merge:                                            ; preds = %while1
  %i9 = load i32, i32* %i
  %tmp10 = add i32 %i9, 1
  store i32 %tmp10, i32* %i
  br label %while

merge13:                                          ; preds = %while
  store i32 2, i32* getelementptr inbounds ([20 x [40 x %_tile]], [20 x [40 x %_tile]]* @_board, i9
0, i32 10, i32 30, i32 2)
  %malloccall14 = tail call i8* @malloc(i32 ptrtoint (%player* getelementptr (%player, %player*
null, i32 1) to i32))
  %_ent_ptr15 = bitcast i8* %malloccall14 to %player*
  %_v_ent_ptr16 = bitcast %player* %_ent_ptr15 to i8*
  store i8* %_v_ent_ptr16, i8** getelementptr inbounds ([20 x [40 x %_tile]], [20 x [40 x %_tile]]*
@_board, i9 0, i32 10, i32 30, i32 3)
  %_build17 = call i1 @_build_player(%_tile* getelementptr inbounds ([20 x [40 x %_tile]], [20 x
[40 x %_tile]]* @_board, i9 0, i32 10, i32 30))
  ret i1 false
}

define i1 @_does_player(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  %this2 = load %_tile*, %_tile** %this1
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %this2, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr
  %_ent_ptr = bitcast i8* %_void_ptr to i1*
  %_called = load i1, i1* %_ent_ptr
  %tmp = xor i1 %_called, true
  br i1 %tmp, label %then, label %else11

merge:                                            ; preds = %else11, %merge9
  ret i1 false

then:                                             ; preds = %entry
  %this3 = load %_tile*, %_tile** %this1
  %_tile_fld_ptr4 = getelementptr inbounds %_tile, %_tile* %this3, i32 0, i32 3
  %_void_ptr5 = load i8*, i8** %_tile_fld_ptr4
  %_ent_ptr6 = bitcast i8* %_void_ptr5 to i1*
  store i1 true, i1* %_ent_ptr6
  %this7 = load %_tile*, %_tile** %this1
  %playerMove_result = call i1 @playerMove(%_tile* %this7)
```

```
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([46 x i8], [46 x i8]* @.str.8, i32 0,
i32 0))
  %getKey = call i8 @getKey()
  %tmp8 = icmp eq i8 %getKey, 113
  br i1 %tmp8, label %then10, label %else

merge9:                                           ; preds = %else, %then10
  br label %merge

then10:                                           ; preds = %then
  store i1 true, i1* @_exit
  br label %merge9

else:                                            ; preds = %then
  br label %merge9

else11:                                           ; preds = %entry
  br label %merge
}

define i1 @_does_goblin(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  ret i1 false
}

define i1 @_build_player(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
  store %_tile* %this, %_tile** %this1
  %this2 = load %_tile*, %_tile** %this1
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %this2, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr
  %_ent_ptr = bitcast i8* %_void_ptr to %player*
  %_ent_fld_ptr = getelementptr inbounds %player, %player* %_ent_ptr, i32 0, i32 1
  store i32 5, i32* %_ent_fld_ptr
  ret i1 false
}

define i1 @_build_goblin(%_tile* %this) {
entry:
  %this1 = alloca %_tile*
```

```
  store %_tile* %this, %_tile** %this1
  %this2 = load %_tile*, %_tile** %this1
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %this2, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr
  %_ent_ptr = bitcast i8* %_void_ptr to %goblin*
  %_ent_fld_ptr = getelementptr inbounds %goblin, %goblin* %_ent_ptr, i32 0, i32 1
  store i32 10, i32* %_ent_fld_ptr
  ret i1 false
}

define i1 @moveOrAttack(%_tile* %p, i32 %r, i32 %c) {
entry:
  %p1 = alloca %_tile*
  store %_tile* %p, %_tile** %p1
  %r2 = alloca i32
  store i32 %r, i32* %r2
  %c3 = alloca i32
  store i32 %c, i32* %c3
  %e = alloca %_tile*
  %r4 = load i32, i32* %r2
  %c5 = load i32, i32* %c3
  %_board = getelementptr [20 x [40 x %_tile]], [20 x [40 x %_tile]]* @_board, i9 0, i32 %r4, i32
%c5
  store %_tile* %_board, %_tile** %e
  %e6 = load %_tile*, %_tile** %e
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %e6, i32 0, i32 2
  %_e_type_ptr = load i32, i32* %_tile_fld_ptr
  %_tmp = icmp eq i32 %_e_type_ptr, 0
  br i1 %_tmp, label %then, label %else

merge:                                            ; preds = %merge19, %then
  ret i1 false

then:                                             ; preds = %entry
  %p7 = load %_tile*, %_tile** %p1
  %r8 = load i32, i32* %r2
  %c9 = load i32, i32* %c3
  %_board10 = getelementptr [20 x [40 x %_tile]], [20 x [40 x %_tile]]* @_board, i9 0, i32 %r8,
i32 %c9
  %_tile_fld_ptr11 = getelementptr inbounds %_tile, %_tile* %p7, i32 0, i32 2
  %_tile_fld_ptr12 = getelementptr inbounds %_tile, %_tile* %p7, i32 0, i32 3
  %_typ = load i32, i32* %_tile_fld_ptr11
  %_ptr = load i8*, i8** %_tile_fld_ptr12
```

```
  %_tile_fld_ptr13 = getelementptr inbounds %_tile, %_tile* %_board10, i32 0, i32 2
  %_tile_fld_ptr14 = getelementptr inbounds %_tile, %_tile* %_board10, i32 0, i32 3
  store i32 %_typ, i32* %_tile_fld_ptr13
  store i8* %_ptr, i8** %_tile_fld_ptr14
  store i32 0, i32* %_tile_fld_ptr11
  store i8* null, i8** %_tile_fld_ptr12
  br label %merge

else:                                             ; preds = %entry
  %e15 = load %_tile*, %_tile** %e
  %_tile_fld_ptr16 = getelementptr inbounds %_tile, %_tile* %e15, i32 0, i32 2
  %_e_type_ptr17 = load i32, i32* %_tile_fld_ptr16
  %_tmp18 = icmp eq i32 %_e_type_ptr17, 1
  br i1 %_tmp18, label %then20, label %else58

merge19:                                          ; preds = %else58, %merge40
  br label %merge

then20:                                           ; preds = %else
  %e21 = load %_tile*, %_tile** %e
  %e22 = load %_tile*, %_tile** %e
  %_tile_fld_ptr23 = getelementptr inbounds %_tile, %_tile* %e22, i32 0, i32 3
  %_void_ptr = load i8*, i8** %_tile_fld_ptr23
  %_ent_ptr = bitcast i8* %_void_ptr to %goblin*
  %_ent_fld_ptr = getelementptr inbounds %goblin, %goblin* %_ent_ptr, i32 0, i32 1
  %health = load i32, i32* %_ent_fld_ptr
  %p24 = load %_tile*, %_tile** %p1
  %_tile_fld_ptr25 = getelementptr inbounds %_tile, %_tile* %p24, i32 0, i32 3
  %_void_ptr26 = load i8*, i8** %_tile_fld_ptr25
  %_ent_ptr27 = bitcast i8* %_void_ptr26 to %player*
  %_ent_fld_ptr28 = getelementptr inbounds %player, %player* %_ent_ptr27, i32 0, i32 1
  %attack = load i32, i32* %_ent_fld_ptr28
  %tmp = sub i32 %health, %attack
  %_tile_fld_ptr29 = getelementptr inbounds %_tile, %_tile* %e21, i32 0, i32 3
  %_void_ptr30 = load i8*, i8** %_tile_fld_ptr29
  %_ent_ptr31 = bitcast i8* %_void_ptr30 to %goblin*
  %_ent_fld_ptr32 = getelementptr inbounds %goblin, %goblin* %_ent_ptr31, i32 0, i32 1
  store i32 %tmp, i32* %_ent_fld_ptr32
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([26 x i8], [26 x i8]* @.str.17, i32
0, i32 0))
  %e33 = load %_tile*, %_tile** %e
  %_tile_fld_ptr34 = getelementptr inbounds %_tile, %_tile* %e33, i32 0, i32 3
  %_void_ptr35 = load i8*, i8** %_tile_fld_ptr34
```

```
  %_ent_ptr36 = bitcast i8* %_void_ptr35 to %goblin*
  %_ent_fld_ptr37 = getelementptr inbounds %goblin, %goblin* %_ent_ptr36, i32 0, i32 1
  %health38 = load i32, i32* %_ent_fld_ptr37
  %tmp39 = icmp sle i32 %health38, 0
  br i1 %tmp39, label %then41, label %else57

merge40:                                          ; preds = %else57, %then41
  br label %merge19

then41:                                           ; preds = %then20
  %printf42 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([15 x i8], [15 x i8]* @.str.18,
i32 0, i32 0))
  %e43 = load %_tile*, %_tile** %e
  %_tile_fld_ptr44 = getelementptr inbounds %_tile, %_tile* %e43, i32 0, i32 2
  %_tile_fld_ptr45 = getelementptr inbounds %_tile, %_tile* %e43, i32 0, i32 3
  store i32 0, i32* %_tile_fld_ptr44
  store i8* null, i8** %_tile_fld_ptr45
  %_ptr46 = load i8*, i8** %_tile_fld_ptr45
  tail call void @free(i8* %_ptr46)
  %p47 = load %_tile*, %_tile** %p1
  %r48 = load i32, i32* %r2
  %c49 = load i32, i32* %c3
  %_board50 = getelementptr [20 x [40 x %_tile]], [20 x [40 x %_tile]]* @_board, i9 0, i32 %r48,
i32 %c49
  %_tile_fld_ptr51 = getelementptr inbounds %_tile, %_tile* %p47, i32 0, i32 2
  %_tile_fld_ptr52 = getelementptr inbounds %_tile, %_tile* %p47, i32 0, i32 3
  %_typ53 = load i32, i32* %_tile_fld_ptr51
  %_ptr54 = load i8*, i8** %_tile_fld_ptr52
  %_tile_fld_ptr55 = getelementptr inbounds %_tile, %_tile* %_board50, i32 0, i32 2
  %_tile_fld_ptr56 = getelementptr inbounds %_tile, %_tile* %_board50, i32 0, i32 3
  store i32 %_typ53, i32* %_tile_fld_ptr55
  store i8* %_ptr54, i8** %_tile_fld_ptr56
  store i32 0, i32* %_tile_fld_ptr51
  store i8* null, i8** %_tile_fld_ptr52
  br label %merge40

else57:                                           ; preds = %then20
  br label %merge40

else58:                                           ; preds = %else
  br label %merge19
}
```

```
define i1 @playerMove(%_tile* %p) {
entry:
  %p1 = alloca %_tile*
  store %_tile* %p, %_tile** %p1
  %r = alloca i32
  %c = alloca i32
  %k = alloca i8
  %moved = alloca i1
  store i1 false, i1* %moved
  %p2 = load %_tile*, %_tile** %p1
  %_tile_fld_ptr = getelementptr inbounds %_tile, %_tile* %p2, i32 0, i32 0
  %_row = load i32, i32* %_tile_fld_ptr
  store i32 %_row, i32* %r
  %p3 = load %_tile*, %_tile** %p1
  %_tile_fld_ptr4 = getelementptr inbounds %_tile, %_tile* %p3, i32 0, i32 1
  %_col = load i32, i32* %_tile_fld_ptr4
  store i32 %_col, i32* %c
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([33 x i8], [33 x i8]* @.str.21, i32
0, i32 0))
  br label %while

while:                                            ; preds = %merge, %entry
  %moved57 = load i1, i1* %moved
  %tmp58 = xor i1 %moved57, true
  br i1 %tmp58, label %while_body, label %merge59

while_body:                                       ; preds = %while
  %getKey = call i8 @getKey()
  store i8 %getKey, i8* %k
  %k5 = load i8, i8* %k
  %tmp = icmp eq i8 %k5, 119
  %r6 = load i32, i32* %r
  %tmp7 = icmp sgt i32 %r6, 0
  %tmp8 = and i1 %tmp, %tmp7
  br i1 %tmp8, label %then, label %else

merge:                                            ; preds = %merge18, %then
  br label %while

then:                                             ; preds = %while_body
  %c9 = load i32, i32* %c
  %r10 = load i32, i32* %r
  %tmp11 = sub i32 %r10, 1
```

```
  %p12 = load %_tile*, %_tile** %p1
  %moveOrAttack_result = call i1 @moveOrAttack(%_tile* %p12, i32 %tmp11, i32 %c9)
  store i1 true, i1* %moved
  br label %merge

else:                                             ; preds = %while_body
  %k13 = load i8, i8* %k
  %tmp14 = icmp eq i8 %k13, 97
  %c15 = load i32, i32* %c
  %tmp16 = icmp sgt i32 %c15, 0
  %tmp17 = and i1 %tmp14, %tmp16
  br i1 %tmp17, label %then19, label %else25

merge18:                                          ; preds = %merge31, %then19
  br label %merge

then19:                                           ; preds = %else
  %c20 = load i32, i32* %c
  %tmp21 = sub i32 %c20, 1
  %r22 = load i32, i32* %r
  %p23 = load %_tile*, %_tile** %p1
  %moveOrAttack_result24 = call i1 @moveOrAttack(%_tile* %p23, i32 %r22, i32 %tmp21)
  store i1 true, i1* %moved
  br label %merge18

else25:                                           ; preds = %else
  %k26 = load i8, i8* %k
  %tmp27 = icmp eq i8 %k26, 115
  %r28 = load i32, i32* %r
  %rows = load i32, i32* @rows
  %tmp29 = icmp slt i32 %r28, %rows
  %tmp30 = and i1 %tmp27, %tmp29
  br i1 %tmp30, label %then32, label %else38

merge31:                                          ; preds = %merge44, %then32
  br label %merge18

then32:                                           ; preds = %else25
  %c33 = load i32, i32* %c
  %r34 = load i32, i32* %r
  %tmp35 = add i32 %r34, 1
  %p36 = load %_tile*, %_tile** %p1
  %moveOrAttack_result37 = call i1 @moveOrAttack(%_tile* %p36, i32 %tmp35, i32 %c33)
```

```
  store i1 true, i1* %moved
  br label %merge31

else38:                                           ; preds = %else25
  %k39 = load i8, i8* %k
  %tmp40 = icmp eq i8 %k39, 100
  %c41 = load i32, i32* %c
  %cols = load i32, i32* @cols
  %tmp42 = icmp slt i32 %c41, %cols
  %tmp43 = and i1 %tmp40, %tmp42
  br i1 %tmp43, label %then45, label %else51

merge44:                                          ; preds = %merge54, %then45
  br label %merge31

then45:                                           ; preds = %else38
  %c46 = load i32, i32* %c
  %tmp47 = add i32 %c46, 1
  %r48 = load i32, i32* %r
  %p49 = load %_tile*, %_tile** %p1
  %moveOrAttack_result50 = call i1 @moveOrAttack(%_tile* %p49, i32 %r48, i32 %tmp47)
  store i1 true, i1* %moved
  br label %merge44

else51:                                           ; preds = %else38
  %k52 = load i8, i8* %k
  %tmp53 = icmp eq i8 %k52, 113
  br i1 %tmp53, label %then55, label %else56

merge54:                                          ; preds = %else56, %then55
  br label %merge44

then55:                                           ; preds = %else51
  store i1 true, i1* @_exit
  store i1 true, i1* %moved
  br label %merge54

else56:                                           ; preds = %else51
  br label %merge54

merge59:                                          ; preds = %while
  ret i1 true
}
```

```
declare noalias i8* @malloc(i32)

declare void @free(i8*)
```

Test Suites

Combined-tests

```
==> fail-assign1.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    num i;
    bool b;
    i = 42;
    i = 10;
    b = true;
    b = false;
    i = false; /* Fail: assigning a bool to an integer */
    return true;
  }
}

==> fail-assign1.gob.out <==
Fatal error: exception Failure("illegal assignment num = bool in i = false")

==> fail-assign2.gob <==
world[20,40] {
```

```
}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    bool b;
    b = 48; /* Fail: assigning an integer to a bool */
    return true;
  }
}

==> fail-assign2.gob.out <==
Fatal error: exception Failure("illegal assignment bool = num in b = 48")

==> fail-assign3.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i = 1;
    i = myvoid();
```

```
    return true;
  }

  bool myvoid()
  {
    return false;
  }

}

==> fail-assign3.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-dead1.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;
    i = 15;
    return i;
    i = 32; /* Error: code after a return */
  }
}

==> fail-dead1.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-dead2.gob <==
world[20,40] {
```

```
}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;

    {
      i = 15;
      return i;
    }
    i = 32; /* Error: code after a return */
    return true;
  }
}

==> fail-dead2.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-for1.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
```

```
functions {
  bool test(entity p) {
    int i;
    for ( ; true ; ) {} /* OK: Forever */

    for (i = 0 ; i < 10 ; i = i + 1) {
      if (i == 3) return 42;
    }

    for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */
    return true;
  }
}

==> fail-for1.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-for2.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;
    for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean */
    return true;
  }
}

==> fail-for2.gob.out <==
Fatal error: exception Parsing.Parse_error
```

```
==> fail-for3.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;
    for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */
    return true;
  }
}

==> fail-for3.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-for4.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
```

```
    int i;
    for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */
    return true;
  }
}

==> fail-for4.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-for5.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;

    for (i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
    return true;
    }
  }
}

==> fail-for5.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-func1.gob <==
world[20,40] {

}
entities {
```

```
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  num foo() {}

  num bar() {}

  num baz() {}

  bool bar() {} /* Error: duplicate function bar */

  bool test(entity p) {

    return true;
  }
}

==> fail-func1.gob.out <==
Fatal error: exception Failure("duplicate function bar")

==> fail-func2.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
```

```
functions {

  num foo(num a, bool b, num c) { }

  void bar(num a, bool b, num a) {} /* Error: duplicate formal a in bar */



  bool test(entity p) {

    return true;
  }
}

==> fail-func2.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-func3.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {

  int foo(int a, bool b, int c) { }

  void bar(int a, void b, int c) {} /* Error: illegal void formal b */

  bool test(entity p) {

    return true;
  }
}
```

```
==> fail-func3.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-func4.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {

  num foo() {}

  bool bar() {}

  char getKey() {} /* Should not be able to define built-in */

  bool baz() {}

  bool test(entity p) {

    return true;
  }
}

==> fail-func4.gob.out <==
Fatal error: exception Failure("function getKey may not be defined")

==> fail-func5.gob <==
world[20,40] {

}
entities {
```

```
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {

  bool foo(num a, bool b)
  {
    return true;
  }

  bool test(entity p) {
    foo(3, false);
    foo(55);
    return true;
  }
}

==> fail-func5.gob.out <==
Fatal error: exception Failure("expecting 2 arguments in foo(55)")

==> fail-func6.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
```

```
  num foo(int a, bool b)
  {
  }


  bool foo(num a, bool b)   /* This should cause an error. We don't allow redefining functions */

  {
    return true;
  }

  bool test(entity p) {
    foo(true);
    foo(3, false);
    return true;
  }
}

==> fail-func6.gob.out <==
Fatal error: exception Parsing.Parse_error

==> goblin_test_skel <==
world[20,40] {
  place(player, 0, 0);
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-add1.gob <==
world[1,1] {
  print( add(17, 25) );
  exit;
}
entities {}
functions {
  num add(num x, num y)
  {
    return x + y;
  }
```

```
}

==> test-add1.gob.out <==
42

==> test-arith1.gob <==
world[1,1] {
  print(39 + 3);
  exit;
}
entities {}
functions {}

==> test-arith1.gob.out <==
42

==> test-arith2.gob <==
world[1,1] {
  print(1 + 2 * 3 + 4);
  exit;
}
entities {}
functions {}

==> test-arith2.gob.out <==
11

==> test-arith3.gob <==
world[1,1] {
  num a;
  a = 42;
  a = a + 5;
  print(a);
  exit;
}
entities {}
functions {
num foo(num a)
{
  return a;
}
}
```

```
==> test-arith3.gob.out <==
47

==> test-col.gob <==
world[20,40] {
  entity e;
  place(player, 0, 0);
  e = peek(0, 0);
  print(col(e));
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-col.gob.out <==
0

==> test-fib.gob <==
world[1,1] {
  print(fib(0));
  print(fib(1));
  print(fib(2));
  print(fib(3));
  print(fib(4));
  print(fib(5));

  exit;
}
entities {}
functions {
num fib(num x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}
}

==> test-fib.gob.out <==
1
1
2
```

```
3
5
8

==> test-field.gob <==
world[20,40] {
  entity e;
  place(player, 0, 0);
  e = peek(0,0);
  print(e.x);
  exit;
}
entities {
  @:player {
    num x;
    build {
      this.x = 5;
    }
  }
}
functions {}

==> test-field.gob.out <==
5

==> test-field1.gob <==
world[20,40] {
  entity e;
  place(player, 0, 0);
  e = peek(0,0);
  e.x = 2;
  print(e.x);
  exit;
}
entities {
  @:player {
    num x;
    build {
      this.x = 5;
    }
  }
}
functions {}
```

```
==> test-field1.gob.out <==
2

==> test-for1.gob <==
world[1,1] {
  num i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    print(i);
  }
  print(42);

  exit;
}
entities {}
functions {}

==> test-for1.gob.out <==
0
1
2
3
4
42

==> test-func1.gob <==
world[1,1] {
  num a;
  a = add(39, 3);
  print(a);

  exit;
}
entities {}
functions {
num add(num a, num b)
{
  return a + b;
}
}

==> test-func1.gob.out <==
42
```

```
==> test-func2.gob <==
world[1,1] {
  num i;
  i = 1;

  fun(i = 2, i = i+1);

  print(i);

  exit;
}
entities {}
functions {
num fun(num x, num y)
{
  return 0;
}
}

==> test-func2.gob.out <==
2

==> test-func3.gob <==
world[1,1] {
  printem(42,17,192,8);

  exit;
}
entities {}
functions {
bool printem(num a, num b, num c, num d)
{
  print(a);
  print(b);
  print(c);
  print(d);
}
}

==> test-func3.gob.out <==
42
17
```

```
192
8

==> test-func4.gob <==
world[1,1] {
  num d;
  d = add(52, 10);
  print(d);
  exit;
}
entities {}
functions {
num add(num a, num b)
{
  num c;
  c = a + b;
  return c;
}
}

==> test-func4.gob.out <==
62

==> test-func5.gob <==
world[1,1] {
  exit;
}
entities {}
functions {

  num foo(num a)
  {
    return a;
  }
}

==> test-func5.gob.out <==

==> test-func6.gob <==
world[1,1] {
  print(bar(17, false, 25));

  exit;
```

```
}
entities {}
functions {
bool foo() {}

num bar(num a, bool b, num c) { return a + c; }
}

==> test-func6.gob.out <==
42

==> test-func7.gob <==
world[1,1] {
  num a;
  a = foo(73);
  print(a);
  exit;
}
entities {}
functions {
  num foo(num c)
  {
    num b;
    b = c + 42;
    return b;
  }
}

==> test-func7.gob.out <==
115

==> test-func8.gob <==
world[1,1] {
  foo(40);

  exit;
}
entities {}
functions {
bool foo(num a)
{
  print(a + 3);
}
```

```
}

==> test-func8.gob.out <==
43

==> test-gcd.gob <==
world[1,1] {
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));

  exit;
}
entities {}
functions {
num gcd(num a, num b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}
}

==> test-gcd.gob.out <==
2
3
11

==> test-gcd2.gob <==
world[1,1] {
  print(gcd(14,21));
  print(gcd(8,36));
  print(gcd(99,121));
  exit;
}
entities {}
functions {
num gcd(num a, num b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
```

```
  return a;
}
}

==> test-gcd2.gob.out <==
7
4
11

==> test-global1.gob <==
world[1,1] {
  rows = 42;
  cols = 21;
  printa();
  printb();
  incab();
  printa();
  printb();
  exit;
}
entities {}
functions {
bool printa()
{
  print(rows);
}

bool printb()
{
  print(cols);
}

bool incab()
{
  rows = rows + 1;
  cols = cols + 1;
}
}

==> test-global1.gob.out <==
42
21
43
```

```
22

==> test-global2.gob <==
world[1,1] {
  num rows; /* Should hide the global rows */
  rows = 42;
  print(rows);
  exit;
}
entities {}
functions {}

==> test-global2.gob.out <==
42

==> test-global3.gob <==
world[1,1] {
  bool rows;
  bool cols;
  rows = true;
  cols = false;
  printb(rows);
  printb(cols);
  exit;
}
entities {}
functions {}

==> test-global3.gob.out <==
1
0

==> test-hello.gob <==
world[1,1] {
  print(42);
  print(71);
  print(1);
  exit;
}
entities {}
functions {}

==> test-hello.gob.out <==
```

```
42
71
1

==> test-if1.gob <==
world[1,1] {
  if (true) print(42);
  print(17);
  exit;
}
entities {}
functions {}

==> test-if1.gob.out <==
42
17

==> test-if2.gob <==
world[1,1] {
  if (true) print(42); else print(8);
  print(17);
  exit;
}
entities {}
functions {}

==> test-if2.gob.out <==
42
17

==> test-if3.gob <==
world[1,1] {
  if (false) print(42);
  print(17);
  exit;
}
entities {}
functions {}

==> test-if3.gob.out <==
17

==> test-if4.gob <==
```

```
world[1,1] {
  if (false) print(42); else print(8);
  print(17);
  exit;
}
entities {}
functions {}

==> test-if4.gob.out <==
8
17

==> test-if5.gob <==
world[1,1] {
  print(cond(true));
  print(cond(false));
  exit;
}
entities {}
functions {
num cond(bool b)
{
  num x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

}

==> test-if5.gob.out <==
42
17

==> test-is.gob <==
world[20,40] {
  if(peek(0,0) is empty) {
    place(player, 0, 0);
  }
  if(peek(0,0) is empty) {
    print(5);
```

```
  } else {
    print(1);
  }
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-is.gob.out <==
1

==> test-local1.gob <==
world[1,1] {
  foo(true);
  exit;
}
entities {}
functions {
bool foo(bool i)
{
  num i; /* Should hide the formal i */

  i = 42;
  print(i + i);
}
}

==> test-local1.gob.out <==
84

==> test-local2.gob <==
world[1,1] {
  print(foo(37, false));
  exit;
}
entities {}
functions {
num foo(num a, bool b)
{
  num c;
  bool d;
```

```
  c = a;

  return c + 10;
}
}

==> test-local2.gob.out <==
47

==> test-move.gob <==
world[20,40] {
  place(player, 0, 0);
  move(peek(0, 0), 1, 1);
  if(peek(1,1) is player) {
    print(1);
  }
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-move.gob.out <==
1

==> test-ops1.gob <==
world[1,1] {
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  printb(1 == 2);
  printb(1 == 1);
  print(99);
  printb(1 != 2);
  printb(1 != 1);
  print(99);
  printb(1 < 2);
  printb(2 < 1);
  print(99);
```

```
  printb(1 <= 2);
  printb(1 <= 1);
  printb(2 <= 1);
  print(99);
  printb(1 > 2);
  printb(2 > 1);
  print(99);
  printb(1 >= 2);
  printb(1 >= 1);
  printb(2 >= 1);
  exit;
}
entities {}
functions {}

==> test-ops1.gob.out <==
3
-1
2
50
99
0
1
99
1
0
99
1
0
99
1
1
0
99
0
1
99
0
1
1

==> test-ops2.gob <==
world[1,1] {
```

```
  printb(true);
  printb(false);
  printb(true and true);
  printb(true and false);
  printb(false and true);
  printb(false and false);
  printb(true or true);
  printb(true or false);
  printb(false or true);
  printb(false or false);
  printb(not false);
  printb(not true);
  print(-10);
  print(--42);
  exit;
}
entities {}
functions {}

==> test-ops2.gob.out <==
1
0
1
0
0
0
1
1
1
0
1
0
-10
42

==> test-peek.gob <==
world[20,40] {
  entity e;
  place(player, 0, 0);
  peek(0, 0);
  peek(1, 1);
  e = peek(0,0);
  e = peek(1,1);
```

```
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-peek.gob.out <==

==> test-place.gob <==
world[20,40] {
  place(player, 0, 0);
  if(peek(0,0) is player) {
    print(5);
  } else {
    print(1);
  }
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-place.gob.out <==
5

==> test-remove.gob <==
world[20,40] {
  place(player, 0, 0);
  remove(peek(0, 0));
  if(peek(0,0) is empty) {
    print(1);
  }
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-remove.gob.out <==
1
```

```
==> test-row.gob <==
world[20,40] {
  entity e;
  place(player, 0, 0);
  e = peek(0, 0);
  print(row(e));
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-row.gob.out <==
0

==> test-var1.gob <==
world[1,1] {
  num a;
  a = 42;
  print(a);
  exit;
}
entities {}
functions {}

==> test-var1.gob.out <==
42

==> test-while1.gob <==
world[1,1] {
  num i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  exit;
}
entities {}
functions {}
```

```
==> test-while1.gob.out <==
5
4
3
2
1
42

==> test-while2.gob <==
world[1,1] {
  print(foo(7));
  exit;
}
entities {}
functions {
num foo(num a)
{
  num j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}
}

==> test-while2.gob.out <==
14

==> fail-assign1.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
```

```
  }

}
functions {
  bool test(entity p) {
    num i;
    bool b;
    i = 42;
    i = 10;
    b = true;
    b = false;
    i = false; /* Fail: assigning a bool to an integer */
    return true;
  }
}

==> fail-assign1.gob.out <==
Fatal error: exception Failure("illegal assignment num = bool in i = false")

==> fail-assign2.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    bool b;
    b = 48; /* Fail: assigning an integer to a bool */
    return true;
  }
}

==> fail-assign2.gob.out <==
```

```
Fatal error: exception Failure("illegal assignment bool = num in b = 48")

==> fail-assign3.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i = 1;
    i = myvoid();
    return true;
  }

  bool myvoid()
  {
    return false;
  }

}

==> fail-assign3.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-dead1.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
```

```
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;
    i = 15;
    return i;
    i = 32; /* Error: code after a return */
  }
}

==> fail-dead1.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-dead2.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;

    {
      i = 15;
      return i;
    }
    i = 32; /* Error: code after a return */
    return true;
```

```
  }
}

==> fail-dead2.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-for1.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;
    for ( ; true ; ) {} /* OK: Forever */

    for (i = 0 ; i < 10 ; i = i + 1) {
      if (i == 3) return 42;
    }

    for (j = 0; i < 10 ; i = i + 1) {} /* j undefined */
    return true;
  }
}

==> fail-for1.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-for2.gob <==
world[20,40] {

}
entities {
```

```
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;
    for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean */
    return true;
  }
}

==> fail-for2.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-for3.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;
    for (i = 0; j < 10 ; i = i + 1) {} /* j undefined */
    return true;
  }
}
```

```
==> fail-for3.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-for4.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  bool test(entity p) {
    int i;
    for (i = 0; i < 10 ; i = j + 1) {} /* j undefined */
    return true;
  }
}

==> fail-for4.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-for5.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }
```

```
}
functions {
  bool test(entity p) {
    int i;

    for (i = 0; i < 10 ; i = i + 1) {
    foo(); /* Error: no function foo */
    return true;
    }
  }
}

==> fail-for5.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-func1.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {
  num foo() {}

  num bar() {}

  num baz() {}

  bool bar() {} /* Error: duplicate function bar */

  bool test(entity p) {

    return true;
```

```
  }
}

==> fail-func1.gob.out <==
Fatal error: exception Failure("duplicate function bar")

==> fail-func2.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {

  num foo(num a, bool b, num c) { }

  void bar(num a, bool b, num a) {} /* Error: duplicate formal a in bar */



  bool test(entity p) {

    return true;
  }
}

==> fail-func2.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-func3.gob <==
world[20,40] {

}
entities {
```

```
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {

  int foo(int a, bool b, int c) { }

  void bar(int a, void b, int c) {} /* Error: illegal void formal b */

  bool test(entity p) {

    return true;
  }
}

==> fail-func3.gob.out <==
Fatal error: exception Parsing.Parse_error

==> fail-func4.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {

  num foo() {}
```

```
  bool bar() {}

  char getKey() {} /* Should not be able to define built-in */

  bool baz() {}

  bool test(entity p) {

    return true;
  }
}

==> fail-func4.gob.out <==
Fatal error: exception Failure("function getKey may not be defined")

==> fail-func5.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {

  bool foo(num a, bool b)
  {
    return true;
  }

  bool test(entity p) {
    foo(3, false);
    foo(55);
    return true;
  }
```

```
}

==> fail-func5.gob.out <==
Fatal error: exception Failure("expecting 2 arguments in foo(55)")

==> fail-func6.gob <==
world[20,40] {

}
entities {
  @:player {
    build {
    }
    does {
      test(this);
      exit;
    }
  }

}
functions {

  num foo(int a, bool b)
  {
  }


  bool foo(num a, bool b)   /* This should cause an error. We don't allow redefining functions */

  {
    return true;
  }

  bool test(entity p) {
    foo(true);
    foo(3, false);
    return true;
  }
}

==> fail-func6.gob.out <==
Fatal error: exception Parsing.Parse_error
```

```
==> goblin_test_skel <==
world[20,40] {
  place(player, 0, 0);
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-add1.gob <==
world[1,1] {
  print( add(17, 25) );
  exit;
}
entities {}
functions {
  num add(num x, num y)
  {
    return x + y;
  }
}

==> test-add1.gob.out <==
42

==> test-arith1.gob <==
world[1,1] {
  print(39 + 3);
  exit;
}
entities {}
functions {}

==> test-arith1.gob.out <==
42

==> test-arith2.gob <==
world[1,1] {
  print(1 + 2 * 3 + 4);
  exit;
}
entities {}
```

```
functions {}

==> test-arith2.gob.out <==
11

==> test-arith3.gob <==
world[1,1] {
  num a;
  a = 42;
  a = a + 5;
  print(a);
  exit;
}
entities {}
functions {
num foo(num a)
{
  return a;
}
}

==> test-arith3.gob.out <==
47

==> test-col.gob <==
world[20,40] {
  entity e;
  place(player, 0, 0);
  e = peek(0, 0);
  print(col(e));
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-col.gob.out <==
0

==> test-fib.gob <==
world[1,1] {
  print(fib(0));
```

```
  print(fib(1));
  print(fib(2));
  print(fib(3));
  print(fib(4));
  print(fib(5));

  exit;
}
entities {}
functions {
num fib(num x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}
}

==> test-fib.gob.out <==
1
1
2
3
5
8

==> test-field.gob <==
world[20,40] {
  entity e;
  place(player, 0, 0);
  e = peek(0,0);
  print(e.x);
  exit;
}
entities {
  @:player {
    num x;
    build {
      this.x = 5;
    }
  }
}
functions {}
```

```
==> test-field.gob.out <==
5

==> test-field1.gob <==
world[20,40] {
  entity e;
  place(player, 0, 0);
  e = peek(0,0);
  e.x = 2;
  print(e.x);
  exit;
}
entities {
  @:player {
    num x;
    build {
      this.x = 5;
    }
  }
}
functions {}

==> test-field1.gob.out <==
2

==> test-for1.gob <==
world[1,1] {
  num i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    print(i);
  }
  print(42);

  exit;
}
entities {}
functions {}

==> test-for1.gob.out <==
0
1
2
3
```

```
4
42

==> test-func1.gob <==
world[1,1] {
  num a;
  a = add(39, 3);
  print(a);

  exit;
}
entities {}
functions {
num add(num a, num b)
{
  return a + b;
}
}

==> test-func1.gob.out <==
42

==> test-func2.gob <==
world[1,1] {
  num i;
  i = 1;

  fun(i = 2, i = i+1);

  print(i);

  exit;
}
entities {}
functions {
num fun(num x, num y)
{
  return 0;
}
}

==> test-func2.gob.out <==
2
```

```
==> test-func3.gob <==
world[1,1] {
  printem(42,17,192,8);

  exit;
}
entities {}
functions {
bool printem(num a, num b, num c, num d)
{
  print(a);
  print(b);
  print(c);
  print(d);
}
}

==> test-func3.gob.out <==
42
17
192
8

==> test-func4.gob <==
world[1,1] {
  num d;
  d = add(52, 10);
  print(d);
  exit;
}
entities {}
functions {
num add(num a, num b)
{
  num c;
  c = a + b;
  return c;
}
}

==> test-func4.gob.out <==
62
```

```
==> test-func5.gob <==
world[1,1] {
  exit;
}
entities {}
functions {

  num foo(num a)
  {
    return a;
  }
}

==> test-func5.gob.out <==

==> test-func6.gob <==
world[1,1] {
  print(bar(17, false, 25));

  exit;
}
entities {}
functions {
bool foo() {}

num bar(num a, bool b, num c) { return a + c; }
}

==> test-func6.gob.out <==
42

==> test-func7.gob <==
world[1,1] {
  num a;
  a = foo(73);
  print(a);
  exit;
}
entities {}
functions {
  num foo(num c)
  {
```

```
    num b;
    b = c + 42;
    return b;
  }
}

==> test-func7.gob.out <==
115

==> test-func8.gob <==
world[1,1] {
  foo(40);

  exit;
}
entities {}
functions {
bool foo(num a)
{
  print(a + 3);
}
}

==> test-func8.gob.out <==
43

==> test-gcd.gob <==
world[1,1] {
  print(gcd(2,14));
  print(gcd(3,15));
  print(gcd(99,121));

  exit;
}
entities {}
functions {
num gcd(num a, num b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}
```

```
}

==> test-gcd.gob.out <==
2
3
11

==> test-gcd2.gob <==
world[1,1] {
  print(gcd(14,21));
  print(gcd(8,36));
  print(gcd(99,121));
  exit;
}
entities {}
functions {
num gcd(num a, num b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}
}

==> test-gcd2.gob.out <==
7
4
11

==> test-global1.gob <==
world[1,1] {
  rows = 42;
  cols = 21;
  printa();
  printb();
  incab();
  printa();
  printb();
  exit;
}
entities {}
functions {
```

```
bool printa()
{
  print(rows);
}

bool printb()
{
  print(cols);
}

bool incab()
{
  rows = rows + 1;
  cols = cols + 1;
}
}

==> test-global1.gob.out <==
42
21
43
22

==> test-global2.gob <==
world[1,1] {
  num rows; /* Should hide the global rows */
  rows = 42;
  print(rows);
  exit;
}
entities {}
functions {}

==> test-global2.gob.out <==
42

==> test-global3.gob <==
world[1,1] {
  bool rows;
  bool cols;
  rows = true;
  cols = false;
  printb(rows);
```

```
  printb(cols);
  exit;
}
entities {}
functions {}

==> test-global3.gob.out <==
1
0

==> test-hello.gob <==
world[1,1] {
  print(42);
  print(71);
  print(1);
  exit;
}
entities {}
functions {}

==> test-hello.gob.out <==
42
71
1

==> test-if1.gob <==
world[1,1] {
  if (true) print(42);
  print(17);
  exit;
}
entities {}
functions {}

==> test-if1.gob.out <==
42
17

==> test-if2.gob <==
world[1,1] {
  if (true) print(42); else print(8);
  print(17);
  exit;
```

```
}
entities {}
functions {}

==> test-if2.gob.out <==
42
17

==> test-if3.gob <==
world[1,1] {
  if (false) print(42);
  print(17);
  exit;
}
entities {}
functions {}

==> test-if3.gob.out <==
17

==> test-if4.gob <==
world[1,1] {
  if (false) print(42); else print(8);
  print(17);
  exit;
}
entities {}
functions {}

==> test-if4.gob.out <==
8
17

==> test-if5.gob <==
world[1,1] {
  print(cond(true));
  print(cond(false));
  exit;
}
entities {}
functions {
num cond(bool b)
{
```

```
  num x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

}

==> test-if5.gob.out <==
42
17

==> test-is.gob <==
world[20,40] {
  if(peek(0,0) is empty) {
    place(player, 0, 0);
  }
  if(peek(0,0) is empty) {
    print(5);
  } else {
    print(1);
  }
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-is.gob.out <==
1

==> test-local1.gob <==
world[1,1] {
  foo(true);
  exit;
}
entities {}
functions {
bool foo(bool i)
{
```

```
  num i; /* Should hide the formal i */

  i = 42;
  print(i + i);
}
}

==> test-local1.gob.out <==
84

==> test-local2.gob <==
world[1,1] {
  print(foo(37, false));
  exit;
}
entities {}
functions {
num foo(num a, bool b)
{
  num c;
  bool d;

  c = a;

  return c + 10;
}
}

==> test-local2.gob.out <==
47

==> test-move.gob <==
world[20,40] {
  place(player, 0, 0);
  move(peek(0, 0), 1, 1);
  if(peek(1,1) is player) {
    print(1);
  }
  exit;
}
entities {
  @:player {}
}
```

```
functions {}

==> test-move.gob.out <==
1

==> test-ops1.gob <==
world[1,1] {
  print(1 + 2);
  print(1 - 2);
  print(1 * 2);
  print(100 / 2);
  print(99);
  printb(1 == 2);
  printb(1 == 1);
  print(99);
  printb(1 != 2);
  printb(1 != 1);
  print(99);
  printb(1 < 2);
  printb(2 < 1);
  print(99);
  printb(1 <= 2);
  printb(1 <= 1);
  printb(2 <= 1);
  print(99);
  printb(1 > 2);
  printb(2 > 1);
  print(99);
  printb(1 >= 2);
  printb(1 >= 1);
  printb(2 >= 1);
  exit;
}
entities {}
functions {}

==> test-ops1.gob.out <==
3
-1
2
50
99
0
```

```
1
99
1
0
99
1
0
99
1
1
0
99
0
1
99
0
1
1

==> test-ops2.gob <==
world[1,1] {
  printb(true);
  printb(false);
  printb(true and true);
  printb(true and false);
  printb(false and true);
  printb(false and false);
  printb(true or true);
  printb(true or false);
  printb(false or true);
  printb(false or false);
  printb(not false);
  printb(not true);
  print(-10);
  print(--42);
  exit;
}
entities {}
functions {}

==> test-ops2.gob.out <==
1
0
```

```
1
0
0
0
1
1
1
0
1
0
-10
42

==> test-peek.gob <==
world[20,40] {
  entity e;
  place(player, 0, 0);
  peek(0, 0);
  peek(1, 1);
  e = peek(0,0);
  e = peek(1,1);
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-peek.gob.out <==

==> test-place.gob <==
world[20,40] {
  place(player, 0, 0);
  if(peek(0,0) is player) {
    print(5);
  } else {
    print(1);
  }
  exit;
}
entities {
  @:player {}
}
```

```
functions {}

==> test-place.gob.out <==
5

==> test-remove.gob <==
world[20,40] {
  place(player, 0, 0);
  remove(peek(0, 0));
  if(peek(0,0) is empty) {
    print(1);
  }
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-remove.gob.out <==
1

==> test-row.gob <==
world[20,40] {
  entity e;
  place(player, 0, 0);
  e = peek(0, 0);
  print(row(e));
  exit;
}
entities {
  @:player {}
}
functions {}

==> test-row.gob.out <==
0

==> test-var1.gob <==
world[1,1] {
  num a;
  a = 42;
  print(a);
```

```
  exit;
}
entities {}
functions {}

==> test-var1.gob.out <==
42

==> test-while1.gob <==
world[1,1] {
  num i;
  i = 5;
  while (i > 0) {
    print(i);
    i = i - 1;
  }
  print(42);
  exit;
}
entities {}
functions {}

==> test-while1.gob.out <==
5
4
3
2
1
42

==> test-while2.gob <==
world[1,1] {
  print(foo(7));
  exit;
}
entities {}
functions {
num foo(num a)
{
  num j;
  j = 0;
  while (a > 0) {
    j = j + 2;
```

```
    a = a - 1;
  }
  return j;
}
}

==> test-while2.gob.out <==
14
```

Test Cases

For each feature, we wrote a test that was supposed to pass and supposed to fail. While a feature was still being implemented, we would also use temporary test scripts to test the incomplete state of the feature. Furthermore, we built small unit tests for existing features.

Automation

We used the testing script attached below.

```
#!/bin/bash

eval $(opam config env)

goblinExecPath=./goblin.native
# testOption=$1 #stores the test flag since functions can't see the $1
# vFlag=$2 #stores the -v flag since functions can't see it with $2
pass=0
fail=0
RED='\033[0;31m'
GREEN='\033[0;32m'
CYAN='\033[0;36m'
NC='\033[0m'
errorFile=errors.log
excpTestFlag=0

logFile=./tests/compiler_tests.log
testPath=./tests/
testExtension=.out
gobbler=./gobble

# Set time limit for all operations
ulimit -t 30


createGoblin(){
```

```
  echo "Creating Goblin compiler"
  make clean 2>&1 > /dev/null
  make 2>&1 > /dev/null
  echo "Compilation of Goblin executable complete"
}

runTests(){
  for testFile in "$testPath"*.gob; do

    filename=$(basename "$testFile")

    echo "==================================" >> session_file
    echo "Testing: $filename" >> session_file

    exe_name=$(basename "$1" | cut -d. -f1)
    if [ -f "$exe_name" ]
    then
      cat $testFile | $goblinExecPath > temp.ll && clang temp.ll 2>&1 > /dev/null
      $gobbler testFile
      ./$exe_name > "$filename".out && rm $exe_name
      #Test output differences use the diff command and neglect screen output
    else

      $gobbler $testFile 2> "$testPath"temp.out
    fi

    diff "$testPath"temp.out "$testPath"$filename$testExtension > /dev/null
    confirmation #function
  done

  rm "$testPath"temp.out

  #Verbose flag actuated
  if [ "$vFlag" == "-v" ]; then
    cat session_file
  fi

  #Copy session output to historical log
  cat session_file >> "$logFile"

  #Test status output
  echo ""
  echo -e "${GREEN}Tests Passed: $pass ${NC}"
```

```
  echo -e "${RED}Tests Failed: $fail ${NC}"
  echo "View $logFile for more information"

  #Clean up temp files
  rm session_file;
}


confirmation(){
  #$? is the exit code for diff, if 0, then test output matched!
  if [ $? -eq 0 ];
  then
    echo -e "${GREEN}$filename passed!${NC}" >> session_file
    echo -e "${GREEN}$filename passed!${NC}"
    ((pass++))

  else
    echo -e "${RED}$filename FAILED${NC}" >> session_file
    echo -e "${RED}$filename FAILED${NC}"

    #print out expected output and result
    echo "Expected Output:" >> session_file

    if [ $excpTestFlag -eq 0 ];   then
      cat "$testPath"$filename$testExtension >> session_file
    else
      cat "$testExceptionsPath"$filename$testExtension >> session_file
    fi
    echo "" >> session_file
    echo "Generated Output:" >> session_file
    cat "$filename".out  >> session_file
    echo "" >> session_file
    ((fail++))
  fi
}


main(){
  createGoblin
  runTests

}
```

main

Contributors

Primarily, Gabe implemented the test suite. Bayard helped.

7. Lessons Learned

Kevin

I learned that a team needs to be punctual and plan early. I also learned that we should break tasks into digestible chunks and iterate towards a minimum viable product. For example, at first, we wanted to implement the world declaration in codegen using a waterfall methodology. We got frustrated. Then, after TA David Watkins' advice, we started developing with a hard-coded array and moving step-by-step. That worked much better.

Bayard

When we started on the project, I would try to add new features, which generally caused some kind of LLVM failure. I did not really understand LLVM so I tried to debug exclusively using the LLVM.moe documentation and the codegen OCaml code. This was a terrible idea which made everything much harder than necessary. Eventually I gave up and started looking through the LLVM output, which was hard at first, but I picked up enough to debug pretty quickly. For future teams I would recommend learning a bit of LLVM at the beginning to make debugging easier.

Gabe

While working on this project, I learned the importance of creating unit tests. When we first created the game loop, the tests we had originally with microc no longer worked. I worked on replacing them with tests that worked with our program structure, but should have probably started sooner than I did. Additionally, as I was the main person that wrote the print helper functions, I learned a lot about the TTY terminal system, how to interact with it using C and its history.

Christina

When we started working on the project, we all worked on the same aspects of the project at the same time. This was not a hindrance until we began working on the more technical aspects (codegen, semantic checker, analyzer, etc), where it would have been more effective to split up the work earlier on than we did. Additionally, we set aside large tasks to complete at a time as opposed to delineating incremental tasks, which made debugging and testing more difficult. Once we properly organized our workload, the remaining tasks were completed more seamlessly.

8. Appendix

goblin.ml by All

```
type action = Ast | LLVM_IR | Compile

let _ =
  let action = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-a", Ast);          (* Print the AST only *)
                              ("-l", LLVM_IR);  (* Generate LLVM, don't check *)
                              ("-c", Compile) ] (* Generate, check LLVM IR *)
  else Compile in
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  let sast = Semant.translate ast in
  Semant.check sast;
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR -> print_string (Llvm.string_of_llmodule (Codegen.translate sast))
  | Compile -> let m = Codegen.translate sast in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

scanner.mll by All

```
{ open Parser
  let unescape s =
    Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
}

let escape = '\\' ['\\' '"' '"' 'n' 'r' 't']
let escape_char = "'" (escape) "'"
let ascii = ([' '-'!' '#'-'[' ']'-'~'])
let digit = ['0'-'9']
let string = '"' ( (ascii | escape)* as s) '"'
let char = "'" ( ascii | digit ) "'"

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| '.'      { DOT }
| '('      { LPAREN }
```

```
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LBRACK }
| ']'      { RBRACK }
| ';'      { SEMI }
| ','      { COMMA }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MOD }
| '='      { ASSIGN }
| "+="     { PLUSAS }
| "-="     { MINUSAS }
| "*="     { TIMESAS }
| "/="     { DIVIDEAS }
| "%="     { MODAS }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| "and"    { AND }
| "or"     { OR }
| "not"    { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "exit"   { EXIT }
| "num"    { NUM }
| "bool"   { BOOL }
| "char"   { CHAR }
| "entity" { ENTITY }
| "true"   { TRUE }
| "false"  { FALSE }
| "does"   { DOES }
| "build"  { BUILD }
| "this"   { THIS }
| "is"     { IS }
```

```
| "empty"  { EMPTY }
| "entities" { ENTITIES }
| "functions" { FUNCTIONS }
| "world" { WORLD }
| (_ as c)':' { SYM(c) }
| ['0'-'9']+ as lxm { NUMLIT(int_of_string lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| char as lxm     { CHARLIT( String.get lxm 1 ) }
| escape_char as lxm{ CHARLIT( String.get (unescape lxm) 1) }
| string          { STRLIT(unescape s) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

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

parser.mly by All

```
%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK COMMA
%token PLUS MINUS TIMES DIVIDE MOD ASSIGN NOT
%token PLUSAS MINUSAS TIMESAS DIVIDEAS MODAS
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR IS
%token RETURN EXIT IF ELSE FOR WHILE
%token NUM BOOL ENTITY EMPTY CHAR
%token DOES BUILD THIS
%token ENTITIES FUNCTIONS WORLD
%token DOT
%token <char> SYM CHARLIT
%token <int> NUMLIT
%token <string> STRLIT
%token <string> ID
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN PLUSAS MINUSAS TIMESAS DIVIDEAS MODAS
%left OR
%left AND
```

```
%left EQ NEQ IS
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD
%right NOT NEG
%left DOT

%start program
%type <Ast.program> program

%%

program:
   world entities functions EOF   { ($1, $2, $3) }

entities:
   ENTITIES LBRACE edecl_list RBRACE { $3 }

edecl_list:
   /* nothing */    { [] }
  | edecl_list edecl { $2 :: $1 }

edecl:
  SYM  ID LBRACE vdecl_list build_opt does_opt RBRACE  { { symbol = $1;
                                                ename = $2;
                                                evars = $4;
                                                ebuild = $5;
                                                edoes = $6 } }

build:
   BUILD LBRACE vdecl_list stmt_list RBRACE { { varlist = $3;
                                        block = List.rev $4; } }

build_opt:
   /* nothing */   { None }
  | build          { Some($1) }

does:
   DOES LBRACE vdecl_list stmt_list RBRACE { { varlist = $3;
                                        block = List.rev $4; } }

does_opt:
  /* nothing */     { None }
```

```
  | does           { Some($1) }

functions:
    FUNCTIONS LBRACE fdecl_list RBRACE { $3 }

fdecl_list:
    /* nothing */    { [] }
  | fdecl_list fdecl { $2 :: $1 }

fdecl:
    ftyp ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
       { { ftyp = $1;
           fname = $2;
           formals = $4;
           locals = List.rev $7;
           body = List.rev $8 } }

formals_opt:
    /* nothing */ { [] }
  | formal_list   { List.rev $1 }

formal_list:
    ftyp ID                   { [($1,$2)] }
  | formal_list COMMA ftyp ID { ($3,$4) :: $1 }

ftyp:
    NUM         { Num }
  | BOOL        { Bool }
  | CHAR        { Char }
  | ENTITY      { Entity }

world:
    WORLD LBRACK NUMLIT COMMA NUMLIT RBRACK LBRACE vdecl_list stmt_list
RBRACE
       { { rows = $3;
           cols = $5;
           wvars = $8;
           wstmts = List.rev $9 } }

vdecl_list:
    /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }
```

```
vdecl:
   NUM ID SEMI                  { (Num, $2) }
  | BOOL ID SEMI                 { (Bool, $2) }
  | CHAR ID SEMI                 { (Char, $2) }
  | ENTITY ID SEMI               { (Entity, $2) }

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) }
  | FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
      { For($3, $5, $7, $9) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
  | EXIT SEMI { Expr (Assign("_exit", BoolLit(true))) }

expr_opt:
   /* nothing */ { BoolLit(false) }
  | expr         { $1 }

expr:
   NUMLIT          { NumLit($1) }
  | CHARLIT         { CharLit($1) }
  | STRLIT          { StrLit($1) }
  | TRUE            { BoolLit(true) }
  | FALSE           { BoolLit(false) }
  | ID              { Id($1) }
  | THIS            { Id("this") }
  | expr DOT ID      { Field($1, $3) }
  | expr IS     ID   { Binop($1, Is, Id($3))}
  | expr IS     EMPTY{ Binop($1, Is, Id("empty"))}
  | expr PLUS   expr { Binop($1, Add,   $3) }
  | expr MINUS  expr { Binop($1, Sub,   $3) }
  | expr TIMES  expr { Binop($1, Mult,  $3) }
  | expr DIVIDE expr { Binop($1, Div,   $3) }
  | expr MOD    expr { Binop($1, Mod,   $3) }
  | expr EQ     expr { Binop($1, Equal, $3) }
  | expr NEQ    expr { Binop($1, Neq,   $3) }
```

```
  | expr LT     expr { Binop($1, Less,  $3) }
  | expr LEQ    expr { Binop($1, Leq,   $3) }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3) }
  | expr AND    expr { Binop($1, And,   $3) }
  | expr OR     expr { Binop($1, Or,    $3) }
  | MINUS expr %prec NEG { Unop(Neg, $2) }
  | NOT expr         { Unop(Not, $2) }
  | ID ASSIGN expr   { Assign($1, $3) }
  | ID PLUSAS expr   { Assign($1, Binop(Id($1), Add, $3)) }
  | ID MINUSAS expr  { Assign($1, Binop(Id($1), Sub, $3)) }
  | ID TIMESAS expr  { Assign($1, Binop(Id($1), Mult, $3)) }
  | ID DIVIDEAS expr { Assign($1, Binop(Id($1), Div, $3)) }
  | ID MODAS expr    { Assign($1, Binop(Id($1), Mod, $3)) }
  | expr DOT ID ASSIGN expr   { FAssign($1, $3, $5) }
  | expr DOT ID PLUSAS expr   { FAssign($1, $3, Binop(Field($1, $3), Add, $5)) }
  | expr DOT ID MINUSAS expr  { FAssign($1, $3, Binop(Field($1, $3), Sub, $5)) }
  | expr DOT ID TIMESAS expr  { FAssign($1, $3, Binop(Field($1, $3), Mult, $5)) }
  | expr DOT ID DIVIDEAS expr { FAssign($1, $3, Binop(Field($1, $3), Div, $5)) }
  | expr DOT ID MODAS expr    { FAssign($1, $3, Binop(Field($1, $3), Mod, $5)) }
  | call             { $1 }
  | LPAREN expr RPAREN { $2 }

call:
   ID LPAREN actuals_opt RPAREN { Call($1, $3) }

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

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

ast.ml by All

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

type uop = Neg | Not

type typ = Num | String | Bool | Char | Entity | Board of int * int
```

```
type expr =
    NumLit of int
  | StrLit of string
  | CharLit of char
  | BoolLit of bool
  | Id of string
  | Field of expr * string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | FAssign of expr * string * expr
  | Call of string * expr list
  | Does of expr

type vdecl = typ * string

type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

type body = {
  varlist : vdecl list;
  block : stmt list;
}

type edecl = {
  symbol : char;
  ename : string;
  evars : vdecl list;
  ebuild : body option;
  edoes : body option;
}

type fdecl = {
  ftyp : typ;
  fname : string;
  formals : vdecl list;
  locals : vdecl list;
  body : stmt list;
```

```
}

type world = {
  rows : int;
  cols : int;
  wvars : vdecl list;
  wstmts : stmt list;
}

type program = world * edecl list * fdecl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "and"
  | Or -> "or"
  | Is -> "is"

let string_of_uop = function
    Neg -> "-"
  | Not -> "not "

let rec string_of_expr = function
    NumLit(l) -> string_of_int l
  | StrLit(l) -> l
  | CharLit(l) -> String.make 1 l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | Id(s) -> s
  | Field(l, s) ->  string_of_expr l ^ "." ^ s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
```

```
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | FAssign(e, f, v) -> string_of_expr e ^ "." ^ f ^ " = " ^ string_of_expr v
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Does(e) -> "DOES(" ^ string_of_expr e ^ ")"

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
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s

let string_of_typ = function
    Num -> "num"
  | Bool -> "bool"
  | Char -> "char"
  | String -> "string"
  | Entity -> "entity"
  | Board (_,_) -> ""

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_edecl edecl =
  String.make 1 edecl.symbol ^ ":" ^
  edecl.ename ^ " {\n" ^
  String.concat "" (List.map string_of_vdecl edecl.evars) ^ "\nbuild {" ^
  String.concat "" (List.map string_of_vdecl (match edecl.ebuild with
  Some(a) -> a.varlist | None -> [])) ^
  String.concat "" (List.map string_of_stmt (match edecl.ebuild with
  Some(a) -> a.block | None -> [])) ^ "}\nbuild {\n" ^
  String.concat "" (List.map string_of_vdecl (match edecl.edoes with
  Some(a) -> a.varlist | None -> [])) ^
  String.concat "" (List.map string_of_stmt (match edecl.edoes with
  Some(a) -> a.block | None -> [])) ^ "}\ndoes\n"
```

```
let string_of_fdecl fdecl =
  string_of_typ fdecl.ftyp ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_world world =
  String.concat ("world [" ^ string_of_int world.rows ^ ", " ^ string_of_int world.cols ^ "] {\n")
  (List.map string_of_vdecl world.wvars) ^ "\n" ^
  String.concat "" (List.map string_of_stmt world.wstmts) ^ "\n}\n"

let string_of_program (w, e, f) =
  String.concat ((string_of_world w) ^ "entities {\n") (List.map string_of_edecl e) ^ "}\n" ^
  String.concat "functions {\n" (List.map string_of_fdecl f) ^ "}\n"
```

semant.ml by Christina, Bayard & Kevin

```
open Ast

module StringMap = Map.Make(String)

 (****************** Translate to Structs **********************)
let translate (world, entities, functions) =

  if not (List.mem "player" (List.map (fun e-> e.ename) entities))
  then raise (Failure ("Player not found")) else ();
  let init_world = {
    ftyp = Bool;
    fname = "_init_world";
    formals = [];
    locals = world.wvars;
    body = world.wstmts
  } in
  let edecl_build l e = (match e.ebuild with
      Some(b) -> { ftyp = Bool;
                   fname = "_build_" ^ e.ename;
                   formals = [(Entity, "this")];
                   locals = b.varlist;
                   body = b.block;
      } :: l
    | None -> { ftyp = Bool;
```

```
            fname = "_build_" ^ e.ename;
            formals = [(Entity, "this")];
            locals = [];
            body = [];
    } :: l)
in
let edecl_does l e = (match e.edoes with
    Some(d) -> { ftyp = Bool;
                 fname = "_does_" ^ e.ename;
                 formals = [(Entity, "this")];
                 locals = d.varlist;
                 body = [
                   If(
                     Unop(Not, Field(Id("this"), "_called")),
                     Block(Expr(FAssign(Id("this"), "_called", BoolLit(true))) :: d.block),
                     Expr(BoolLit(false))
                   )
                 ];
    } :: l
  | None -> { ftyp = Bool;
            fname = "_does_" ^ e.ename;
            formals = [(Entity, "this")];
            locals = [];
            body = [];
    } :: l)
in
let functions = List.fold_left edecl_build functions entities in
let functions = List.fold_left edecl_does functions entities in
let functions = init_world :: functions in
let nothing = {
  ftyp = Bool;
  fname = "_nothing";
  formals = [(Entity, "this")];
  locals = [];
  body = []
} in
let main = {
  ftyp = Num;
  fname = "main";
  formals = [];
  locals = [(Num, "i"); (Num, "j"); (Entity, "e")];
  body = [
    Expr(Assign("rows", NumLit(world.rows)));
```

```
Expr(Assign("cols", NumLit(world.cols)));
For(Assign("i", NumLit(0)),
   Binop(Id("i"), Less, Id("rows")),
   Assign("i", Binop(Id("i"), Add, NumLit(1))),

   For(Assign("j", NumLit(0)),
      Binop(Id("j"), Less, Id("cols")),
      Assign("j", Binop(Id("j"), Add, NumLit(1))),
      Expr(Call("_init_tile", [Id("i"); Id("j")]))
   )
);
Expr(Call("_init_world", []));
While(Unop(Not, Id("_exit")), Block([
  Expr(Call("_clearScreen", []));
  For(
    Assign("i", NumLit(0)),
    Binop(Id("i"), Less, Id("rows")),
    Assign("i", Binop(Id("i"), Add, NumLit(1))),
    Block([
      For(
        Assign("j", NumLit(0)),
        Binop(Id("j"), Less, Id("cols")),
        Assign("j", Binop(Id("j"), Add, NumLit(1))),
        Block([
          Expr(Assign("e", Call("peek", [Id("i"); Id("j")])));
          If(
            Unop(Not, Binop(Id("e"), Is, Id("empty"))),
            Expr(FAssign(Id("e"), "_called", BoolLit(false))),
            Expr(BoolLit(false))
          );
          Expr(Call("_print_symbol", [Id("e")]));
        ])
      );
      Expr(Call("prints", [StrLit("")]))
    ])
  );
  For(
    Assign("i", NumLit(0)),
    Binop(Id("i"), Less, Id("rows")),
    Assign("i", Binop(Id("i"), Add, NumLit(1))),
    Block([
      For(
        Assign("j", NumLit(0)),
```

```
            Binop(Id("j"), Less, Id("cols")),
            Assign("j", Binop(Id("j"), Add, NumLit(1))),
            Block([
              Expr(Assign("e", Call("peek", [Id("i"); Id("j")])));
              Expr(Does(Id("e")));
            ])
          );
        ])
      )
    ]));
    Return(NumLit(0))
  ]
} in
let functions = nothing :: functions in
let functions = main :: functions in
let edecl_struct l e = (e.ename, List.rev ((Bool, "_called") :: (List.rev e.evars))) :: l in
let structs = List.fold_left edecl_struct [] entities in
let edecl_symbol l e = (e.ename, e.symbol) :: l in
let symbols = List.fold_left edecl_symbol [] entities in
([
  (Board(world.rows, world.cols), "_board");
  (Num, "rows");
  (Num, "cols");
  (Bool, "_exit")
], functions, structs, symbols)


(* Semantic checking of a program. Returns void if successful,
   throws an exception if something is wrong.

   Check each global variable, function, entity and world. *)


let check (globals, functions, structs, _) =

  (*********************** Exceptions ***********************************)

  (* Raise an exception if the given list has a duplicate *)
  let report_duplicate exceptf list =
    let rec helper = function
  n1 :: n2 :: _ when n1 = n2 -> raise (Failure (exceptf n1))
      | _ :: t -> helper t
      | [] -> ()
```

```
  in helper (List.sort compare list)
in


(* Raise an exception of the given rvalue type cannot be assigned to the given lvalue type *)
let check_assign lvaluet rvaluet err =
  if lvaluet == rvaluet then lvaluet else raise err
in

(****************** Checking Global Variables *****************************)

report_duplicate (fun n -> "duplicate global " ^ n) (List.map snd globals);

 (************************ Structs ***************************************)
let s_vars = List.map snd structs in
let s_vars = List.flatten s_vars in
let s_vars = List.fold_left (fun l (t, n) -> if n = "_called" then l else (t, n) :: l) [] s_vars in
(* Checks for duplicates *)
report_duplicate (fun s -> "Duplicate attribute. " ^ s) (List.map snd s_vars);

let s_name = List.map fst structs in
(* Checks for duplicate struct names. *)
report_duplicate (fun s -> "Duplicate struct name." ^ s) s_name;

let s_map = List.fold_left (fun m n -> StringMap.add n 0 m) StringMap.empty s_name in
let sf_map = List.fold_left (fun m (t, n) -> StringMap.add n t m) StringMap.empty s_vars in
let sf_map = StringMap.add "_called" Bool sf_map in

(************************ Checking Functions ****************************)

if List.mem "print" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function print may not be defined")) else ();

if List.mem "prints" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function prints may not be defined")) else ();

if List.mem "getKey" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function getKey may not be defined")) else ();

if List.mem "row" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function row may not be defined")) else ();

if List.mem "col" (List.map (fun fd -> fd.fname) functions)
```

```
then raise (Failure ("function col may not be defined")) else ();

if List.mem "move" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function move may not be defined")) else ();

if List.mem "peek" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function peek may not be defined")) else ();

if List.mem "remove" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function remove may not be defined")) else ();

if List.mem "place" (List.map (fun fd -> fd.fname) functions)
then raise (Failure ("function place may not be defined")) else ();

report_duplicate (fun n -> "duplicate function " ^ n)
(List.map (fun fd -> fd.fname) functions);

(* Function declaration for a named function *)
let built_in_decls = StringMap.empty in
let built_in_decls = StringMap.add "printb"
   { ftyp = Bool; fname = "printb"; formals = [(Bool, "x")];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "print"
   { ftyp = Bool; fname = "print"; formals = [(Num, "x")];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "_print_symbol"
   { ftyp = Bool; fname = "_print_symbol"; formals = [(Entity, "x")];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "prints"
   { ftyp = Bool; fname = "prints"; formals = [(String, "s")];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "getKey"
   { ftyp = Char; fname = "getKey"; formals = [];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "_clearScreen"
   { ftyp = Num; fname = "_clearScreen"; formals = [];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "row"
   { ftyp = Num; fname = "row"; formals = [(Entity, "x")];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "col"
   { ftyp = Num; fname = "col"; formals = [(Entity, "x")];
     locals = []; body = [] } built_in_decls in
```

```
let built_in_decls = StringMap.add "move"
   { ftyp = Bool; fname = "move"; formals = [(Entity, "x"); (Num, "y"); (Num, "z")];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "peek"
   { ftyp = Entity; fname = "peek"; formals = [(Num, "x"); (Num, "y")];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "remove"
   { ftyp = Bool; fname = "remove"; formals = [(Entity, "x")];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "place"
   { ftyp = Bool; fname = "place"; formals = [];
     locals = []; body = [] } built_in_decls in
let built_in_decls = StringMap.add "_init_tile"
   { ftyp = Bool; fname = "_init_tile"; formals = [(Num, "x"); (Num, "y")];
     locals = []; body = [] } built_in_decls
in

let function_decls = List.fold_left (fun m fd -> StringMap.add fd.fname fd m)
built_in_decls functions
in

let function_decl s = try StringMap.find s function_decls
with Not_found -> raise (Failure ("unrecognized function " ^ s))
in

let check_function func =

  report_duplicate (fun n -> "duplicate formal " ^ n ^ " in " ^ func.fname)
  (List.map snd func.formals);

  report_duplicate (fun n -> "duplicate local " ^ n ^ " in " ^ func.fname)
  (List.map snd func.locals);

  (* Type of each variable (global, formal, or local *)
  let symbols = List.fold_left (fun m (t, n) -> StringMap.add n t m)
    StringMap.empty (globals @ func.formals @ func.locals )
  in

  let type_of_identifier s =
    try StringMap.find s symbols
    with Not_found -> raise (Failure ("undeclared identifier " ^ s))
  in
```

```
   (* Return the type of an expression or throw an exception *)
   let rec expr = function
       NumLit _ -> Num
     | BoolLit _ -> Bool
     | CharLit _ -> Char
     | StrLit _ -> String
     | Id s -> type_of_identifier s
     | Field(_, f) -> if StringMap.mem f sf_map then StringMap.find f sf_map
       else raise (Failure ("undefined field " ^ f))
     | Binop(e, Is, Id s) as ex -> let t = expr e in
       if StringMap.mem s s_map then
         (ignore (check_assign t Entity (Failure ("can only check if an entity is of a certain type,
attempted to check " ^ string_of_typ t  ^ " in " ^ string_of_expr e))); Bool)
       else
         if s = "empty" then Bool else
         raise (Failure ("entity type being checked for not defined in " ^ string_of_expr ex))
     | Binop(e1, op, e2) as e -> let t1 = expr e1 and t2 = expr e2 in
       (match op with
         Add | Sub | Mult | Div when t1 = Num && t2 = Num -> Num
       | Equal | Neq when t1 = t2 -> Bool
       | Less | Leq | Greater | Geq when t1 = Num && t2 = Num -> Bool
       | And | Or when t1 = Bool && t2 = Bool -> Bool
       | _ -> raise (Failure ("illegal binary operator " ^
             string_of_typ t1 ^ " " ^ string_of_op op ^ " " ^
             string_of_typ t2 ^ " in " ^ string_of_expr e))
       )
     | Unop(op, e) as ex -> let t = expr e in
       (match op with
         Neg when t = Num -> Num
       | Not when t = Bool -> Bool
       | _ -> raise (Failure ("illegal unary operator " ^ string_of_uop op ^
             string_of_typ t ^ " in " ^ string_of_expr ex)))
     | Assign(var, e) as ex -> let lt = type_of_identifier var
                               and rt = expr e in
       check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^
                                    " = " ^ string_of_typ rt ^ " in " ^
                                    string_of_expr ex))
     | FAssign(_, f, v) as ex -> let lt = if StringMap.mem f sf_map then StringMap.find f sf_map
else raise (Failure ("undefined field " ^ f)) and rt = expr v in
       check_assign lt rt (Failure ("illegal assignment " ^ string_of_typ lt ^
                                    " = " ^ string_of_typ rt ^ " in " ^
                                    string_of_expr ex))
     | Does _ -> Bool
```

```
  | Call(fname, actuals) as call ->
    (match fname with
      "prints" ->
      (match actuals with
        [StrLit _] -> Bool
      | _ -> raise (Failure ("expecting 1 argument in " ^ string_of_expr call))
      )
    | "place" ->
        if List.length actuals != 3 then
            raise (Failure ("expecting 3 arguments in " ^ string_of_expr call))
        else ();
        (match (List.hd actuals) with
            Id s -> if StringMap.mem s s_map then ()
            else raise (Failure ("first argument must be name of entity in " ^
            string_of_expr call))
          | _ -> raise (Failure ("first argument must be entity type in " ^
            string_of_expr call))
          );
             ignore(check_assign Num (expr (List.nth actuals 1)) (Failure
             ("illegal actual argument found " ^ string_of_typ (expr (List.nth actuals 1)) ^
                 " expected Num ")));
             check_assign Num (expr (List.nth actuals 2)) (Failure
             ("illegal actual argument found " ^ string_of_typ (expr (List.nth actuals 2)) ^
                 " expected Num "))
    | _ ->
      let fd = function_decl fname in
      if List.length actuals != List.length fd.formals then
        raise (Failure ("expecting " ^ string_of_int
        (List.length fd.formals) ^ " arguments in " ^ string_of_expr call))
      else
        List.iter2 (fun (ft, _) e -> let et = expr e in
        ignore (check_assign ft et
             (Failure ("illegal actual argument found " ^ string_of_typ et ^
             " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr e))))
          fd.formals actuals;
          fd.ftyp
      )
in

let check_bool_expr e = if expr e != Bool
  then raise (Failure ("expected Boolean expression in " ^ string_of_expr e))
  else () in
```

```
  (* Verify a statement or throw an exception *)
  let rec stmt = function
    Block sl -> let rec check_block = function
      [Return _ as s] -> stmt s
        | Return _ :: _ -> raise (Failure "nothing may follow a return")
        | Block sl :: ss -> check_block (sl @ ss)
        | s :: ss -> stmt s ; check_block ss
       | [] -> ()
  in check_block sl
       | Expr e -> ignore (expr e)
       | Return e -> let t = expr e in if t = func.ftyp then () else
         raise (Failure ("return gives " ^ string_of_typ t ^ " expected " ^
                         string_of_typ func.ftyp ^ " in " ^ string_of_expr e))

       | If(p, b1, b2) -> check_bool_expr p; stmt b1; stmt b2
    | For(e1, e2, e3, st) -> ignore (expr e1); check_bool_expr e2;
                             ignore (expr e3); stmt st
    | While(p, s) -> check_bool_expr p; stmt s
  in

  stmt (Block func.body)
  in
 List.iter check_function functions
```

codegen.ml by Bayard & Kevin & Gabe

```
module L = Llvm
module A = Ast

module StringMap = Map.Make(String)

type struct_info = {tcode: int; ltype: L.lltype; fields: int StringMap.t}

let translate (globals, functions, structs, symbols) =
  let context = L.global_context () in
  let the_module = L.create_module context "Goblan"
  and i32_t  = L.i32_type  context
  and i8_t   = L.i8_type   context
  and i1_t   = L.i1_type   context
  and ptr_t  = L.pointer_type
  and arr_t  = L.array_type
  and tile_t = L.named_struct_type context "_tile"
  and typedefs =
```

```
   let structs_l = (fun (_, l) -> l) (List.fold_left (fun (c, l) f -> (c - 1, (c, f) :: l)) (List.length structs, [])
structs) in
   let add_struct map (tcode, (name, fields)) =
     let ltype = L.named_struct_type context name in
     let fields_l = (fun (_, l) -> l) (List.fold_left (fun (c, l) f -> (c - 1, (c, f) :: l)) (List.length fields - 1,
[]) fields) in
     let fields_m = List.fold_left (fun m (i, (_, n)) -> StringMap.add n i m) StringMap.empty fields_l
in
     let struct_info = {tcode = tcode; ltype = ltype; fields = fields_m} in
     StringMap.add name struct_info map
   in
   List.fold_left add_struct StringMap.empty structs_l
 in
 let func_t = L.function_type i1_t [|ptr_t tile_t|] in

 let field_info =
   let add_field t m (_, n) = StringMap.add n (t, StringMap.find n (StringMap.find t
typedefs).fields) m in
   let add_struct m (n, l) = List.fold_left (fun m e -> add_field n m e) m l in
   List.fold_left add_struct StringMap.empty structs
 in

 let ltype_of_typ = function
     A.Num -> i32_t
   | A.Entity -> ptr_t tile_t
   | A.Bool -> i1_t
   | A.Char -> i8_t
   | A.String -> ptr_t i8_t
   | A.Board (row, col) -> arr_t (arr_t tile_t col) row
 in

 (* Define tile *)
 ignore(L.struct_set_body tile_t [|i32_t; i32_t; i32_t; ptr_t i8_t|] false);
 (* Define struct *)
 let build_struct_body (name, fields) =
   let types = Array.of_list (List.rev (List.map (fun (t, _) -> ltype_of_typ t) fields)) in
   ignore(L.struct_set_body (StringMap.find name typedefs).ltype types false)
 in
 List.iter build_struct_body structs;


 (* Declare each global variable; remember its value in a map *)
 let global_vars =
```

```
  let global_var m (t, n) =
    let init = (match t with
        A.Board (r, c) -> let z = L.const_int tile_t 0 in
                          let rz = L.const_array tile_t (Array.make c z) in
                          L.const_array (arr_t tile_t c) (Array.make r rz)
      | _ -> L.const_int (ltype_of_typ t) 0)
    in
    StringMap.add n (L.define_global n init the_module) m
  in
  List.fold_left global_var StringMap.empty globals
in

  (* Add "struct type code to symbol representation" array to global vars *)
  let global_vars =
    let n = "_symbols" in
    let sym_arr = Array.make (List.length symbols + 1) (L.const_int i8_t 32) in
    let build_sym (n, c) = sym_arr.((StringMap.find n typedefs).tcode) <- (L.const_int i8_t
(int_of_char c)) in
    List.iter build_sym symbols;
    let init = L.const_array i8_t sym_arr in
    StringMap.add n (L.define_global n init the_module) global_vars
  in

  (* Declare printf(), which the print built-in function will call *)
  let printf_t = L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func = L.declare_function "printf" printf_t the_module in

  let getKey_t = L.function_type i8_t [| |] in
  let getKey_func = L.declare_function "getKey" getKey_t the_module in

  let clearScreen_t = L.function_type i32_t [| |] in
  let clearScreen_func = L.declare_function "_clearScreen" clearScreen_t the_module in

  (* Define each function (arguments and return type) so we can call it *)
  let function_decls =
    let function_decl m fdecl =
      let name = fdecl.A.fname
      and formal_types =
          Array.of_list (List.map (fun (t,_) -> ltype_of_typ t) fdecl.A.formals)
      in let ftype = L.function_type (ltype_of_typ fdecl.A.ftyp) formal_types in
      StringMap.add name (L.define_function name ftype the_module, fdecl) m in
    List.fold_left function_decl StringMap.empty functions in
```

```
(* Add "struct type code to build function pointer" array to global vars *)
let global_vars =
  let n = "_build_tbl" in
  let nothing_func = fst (StringMap.find "_nothing" function_decls) in
  let fn_arr = Array.make (List.length symbols + 1) nothing_func in
  let build_fn (n, _) =
    let tcode = (StringMap.find n typedefs).tcode in
    let fname = "_build_" ^ n in
    fn_arr.(tcode) <- fst (StringMap.find fname function_decls)
  in
  List.iter build_fn structs;
  let init = L.const_array (ptr_t func_t) fn_arr in
  StringMap.add n (L.define_global n init the_module) global_vars
in

(* Add "struct type code to does function pointer" array to global vars *)
let global_vars =
  let n = "_does_tbl" in
  let nothing_func = fst (StringMap.find "_nothing" function_decls) in
  let fn_arr = Array.make (List.length symbols + 1) nothing_func in
  let does_fn (n, _) =
    let tcode = (StringMap.find n typedefs).tcode in
    let fname = "_does_" ^ n in
    fn_arr.(tcode) <- fst (StringMap.find fname function_decls)
  in
  List.iter does_fn structs;
  let init = L.const_array (ptr_t func_t) fn_arr in
  StringMap.add n (L.define_global n init the_module) global_vars
in

(* Fill in the body of the given function *)
let build_function_body fdecl =
  let (the_function, _) = StringMap.find fdecl.A.fname function_decls in
  let builder = L.builder_at_end context (L.entry_block the_function) in

  let int_format_str = L.build_global_stringptr "%d\n" "fmt" builder in
  let char_format_str = L.build_global_stringptr "%c" "fmt" builder in

  (* Construct the function's "locals": formal arguments and locally
     declared variables.  Allocate each on the stack, initialize their
     value, if appropriate, and remember their values in the "locals" map *)
  let local_vars =
    let add_formal m (t, n) p = L.set_value_name n p;
```

```
            let local = L.build_alloca (ltype_of_typ t) n builder in
            ignore (L.build_store p local builder);
            StringMap.add n local m in

  let add_local m (t, n) =
            let local_var = L.build_alloca (ltype_of_typ t) n builder
            in StringMap.add n local_var m in

  let formals = List.fold_left2 add_formal StringMap.empty fdecl.A.formals
      (Array.to_list (L.params the_function)) in
  List.fold_left add_local formals fdecl.A.locals in

(* Return the value for a variable or formal argument *)
let lookup n = try StringMap.find n local_vars
               with Not_found -> StringMap.find n global_vars
in

(* Helper function for accessing a tile of the board *)
let get_tile_ptr (r, c, b) =
  let s = "_board" in
  L.build_gep (lookup s) [|L.const_int tile_t 0; r; c|] s b
in

(* Helper function for accessing a field of a tile that's in the board*)
let get_tile_field_ptr (e, f, b) =
  let idx = (match f with
      "row" -> 0
    | "col" -> 1
    | "type" -> 2
    | _ -> 3)
  in
  L.build_struct_gep e idx "_tile_fld_ptr" b
in

(* Helper function for accessing a field of a struct that's in a tile *)
let get_entity_field_ptr (e, f, b) =
  let tile_field_ptr = get_tile_field_ptr (e, "ptr", b) in
  let void_ptr = L.build_load tile_field_ptr "_void_ptr" b in
  if f = "_called" then
    L.build_bitcast void_ptr (ptr_t i1_t) "_ent_ptr" b
  else
    let (t, idx) = StringMap.find f field_info in
    let lt = (StringMap.find t typedefs).ltype in
```

```
    let ent_ptr = L.build_bitcast void_ptr (ptr_t lt) "_ent_ptr" b in
    L.build_struct_gep ent_ptr idx "_ent_fld_ptr" b
in

(* Construct code for an expression; return its value *)
let rec expr builder = function
        A.NumLit i -> L.const_int i32_t i
  | A.StrLit s -> L.const_string context s
  | A.CharLit c -> L.const_int i8_t (int_of_char c)
  | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
  | A.Id s -> L.build_load (lookup s) s builder
  | A.Field (e, s) ->
      let e' = expr builder e in
      let ptr = get_entity_field_ptr (e', s, builder) in
      L.build_load ptr s builder
  | A.Binop (e, A.Is, A.Id t) ->
      let e' = expr builder e in
      let e_type_ptr = get_tile_field_ptr (e', "type", builder) in
      let e_type = L.build_load e_type_ptr "_e_type_ptr" builder in
      let tcode = if t = "empty" then 0 else
        (StringMap.find t typedefs).tcode in
      L.build_icmp L.Icmp.Eq e_type (L.const_int i32_t tcode) "_tmp" builder
  | A.Binop (e1, op, e2) ->
        let e1' = expr builder e1
        and e2' = expr builder e2 in
        (match op with
          A.Add     -> L.build_add
        | A.Sub     -> L.build_sub
        | A.Mult    -> L.build_mul
    | A.Div     -> L.build_sdiv
    | A.Mod     -> L.build_srem
        | A.And     -> L.build_and
        | A.Or      -> L.build_or
        | A.Equal | A.Is -> L.build_icmp L.Icmp.Eq
        | A.Neq     -> L.build_icmp L.Icmp.Ne
        | A.Less    -> L.build_icmp L.Icmp.Slt
        | A.Leq     -> L.build_icmp L.Icmp.Sle
        | A.Greater -> L.build_icmp L.Icmp.Sgt
        | A.Geq     -> L.build_icmp L.Icmp.Sge
        ) e1' e2' "tmp" builder
  | A.Unop(op, e) ->
        let e' = expr builder e in
        (match op with
```

```
        A.Neg     -> L.build_neg
    | A.Not     -> L.build_not) e' "tmp" builder
| A.Assign (s, e) -> let e' = expr builder e in
                     ignore (L.build_store e' (lookup s) builder); e'
| A.FAssign (e, s, v) ->
    let e' = expr builder e in
    let v' = expr builder v in
    let ptr = get_entity_field_ptr (e', s, builder) in
    ignore (L.build_store v' ptr builder); v'
| A.Call ("print", [e]) | A.Call ("printb", [e]) ->
       L.build_call printf_func [| int_format_str ; (expr builder e) |] "printf" builder
| A.Call ("_print_symbol", [e]) ->
    let e' = expr builder e in
    let typ_ptr = get_tile_field_ptr (e', "type", builder) in
    let tcode = L.build_load typ_ptr "_tcode" builder in
    let arr_ptr = lookup "_symbols" in
    let sym_ptr = L.build_gep arr_ptr [|L.const_int i8_t 0; tcode|] "_sym_ptr" builder in
    let sym = L.build_load sym_ptr "_sym" builder in
       L.build_call printf_func [| char_format_str ; sym |] "printf" builder
| A.Call ("prints", [e]) ->
    let get_string = function A.StrLit s -> s | _ -> "" in
    let s_ptr = L.build_global_stringptr ((get_string e) ^ "\n") ".str" builder in
    L.build_call printf_func [| s_ptr |] "printf" builder
| A.Call ("getKey", []) ->
    L.build_call getKey_func [| |] "getKey" builder
| A.Call ("_clearScreen", []) ->
    L.build_call clearScreen_func [| |] "_clearScreen" builder
| A.Call ("row", [e]) ->
    let e' = expr builder e in
    L.build_load (get_tile_field_ptr (e', "row", builder)) "_row" builder
| A.Call ("col", [e]) ->
    let e' = expr builder e in
    L.build_load (get_tile_field_ptr (e', "col", builder)) "_col" builder
| A.Call ("move", [e; r; c]) ->
    let tile_ptr = expr builder e in
    let r' = expr builder r in
    let c' = expr builder c in
    let new_tile_ptr = get_tile_ptr (r', c', builder) in
    let type_ptr = get_tile_field_ptr (tile_ptr, "type", builder) in
    let ptr_ptr = get_tile_field_ptr (tile_ptr, "ptr", builder) in
    let typ = L.build_load type_ptr "_typ" builder in
    let ptr = L.build_load ptr_ptr "_ptr" builder in
    let new_type_ptr = get_tile_field_ptr (new_tile_ptr, "type", builder) in
```

```
    let new_ptr_ptr = get_tile_field_ptr (new_tile_ptr, "ptr", builder) in
    ignore(L.build_store typ new_type_ptr builder);
    ignore(L.build_store ptr new_ptr_ptr builder);
    ignore(L.build_store (L.const_int i32_t 0) type_ptr builder);
    ignore(L.build_store (L.const_pointer_null (ptr_t i8_t)) ptr_ptr builder);
    L.const_int i1_t 1
  | A.Call ("peek", [r; c]) ->
    let r' = expr builder r in
    let c' = expr builder c in
    get_tile_ptr (r', c', builder)
  | A.Call ("remove", [e]) ->
    let e' = expr builder e in
    let type_ptr = get_tile_field_ptr (e', "type", builder) in
    let ptr_ptr = get_tile_field_ptr (e', "ptr", builder) in
    ignore(L.build_store (L.const_int i32_t 0) type_ptr builder);
    ignore(L.build_store (L.const_pointer_null (ptr_t i8_t)) ptr_ptr builder);
    let ptr = L.build_load ptr_ptr "_ptr" builder in
    L.build_free ptr builder
  | A.Call ("place", [A.Id (t); r; c]) ->
    let r' = expr builder r in
    let c' = expr builder c in
    let tile_ptr = get_tile_ptr (r', c', builder) in
    let tinfo = StringMap.find t typedefs in
    let tcode = L.const_int i32_t tinfo.tcode in
    ignore (L.build_store tcode (get_tile_field_ptr (tile_ptr, "type", builder)) builder);
    let s_ptr = L.build_malloc tinfo.ltype "_ent_ptr" builder in
    let void_s_ptr = L.build_bitcast s_ptr (ptr_t i8_t) "_v_ent_ptr" builder in
    ignore (L.build_store void_s_ptr (get_tile_field_ptr (tile_ptr, "ptr", builder)) builder);
    let fn_ptr = fst (StringMap.find ("_build_" ^ t) function_decls) in
       L.build_call fn_ptr [| tile_ptr |] "_build" builder
  | A.Call ("_init_tile", [r; c]) ->
    let r' = expr builder r in
    let c' = expr builder c in
    let tile_ptr = get_tile_ptr (r', c', builder) in
    ignore (L.build_store r' (get_tile_field_ptr (tile_ptr, "row", builder)) builder);
    ignore (L.build_store c' (get_tile_field_ptr (tile_ptr, "col", builder)) builder);
    L.const_int i1_t 1
  | A.Call (f, act) ->
    let (fdef, _) = StringMap.find f function_decls in
       let actuals = List.rev (List.map (expr builder) (List.rev act)) in
       let result = f ^ "_result" in
    L.build_call fdef (Array.of_list actuals) result builder
  | A.Does e ->
```

```
      let e' = expr builder e in
      let typ_ptr = get_tile_field_ptr (e', "type", builder) in
      let tcode = L.build_load typ_ptr "_tcode" builder in
      let arr_ptr = lookup "_does_tbl" in
      let fn_ptr_ptr = L.build_gep arr_ptr [|L.const_int i8_t 0; tcode|] "_fn_ptr_ptr" builder in
      let fn_ptr = L.build_load fn_ptr_ptr "_fn_ptr" builder in
         L.build_call fn_ptr [| e' |] "_does" builder
in

(* Invoke "f builder" if the current block doesn't already
   have a terminal (e.g., a branch). *)
let add_terminal builder f =
  match L.block_terminator (L.insertion_block builder) with
       Some _ -> ()
  | None -> ignore (f builder) in

(* Build the code for the given statement; return the builder for
   the statement's successor *)
let rec stmt builder = function
       A.Block sl -> List.fold_left stmt builder sl
  | A.Expr e -> ignore (expr builder e); builder
  | A.Return e -> ignore (L.build_ret (expr builder e) builder); builder
  | A.If (predicate, then_stmt, else_stmt) ->
     let bool_val = expr builder predicate in
        let merge_bb = L.append_block context "merge" the_function in

        let then_bb = L.append_block context "then" the_function in
        add_terminal (stmt (L.builder_at_end context then_bb) then_stmt)
          (L.build_br merge_bb);

        let else_bb = L.append_block context "else" the_function in
        add_terminal (stmt (L.builder_at_end context else_bb) else_stmt)
          (L.build_br merge_bb);

        ignore (L.build_cond_br bool_val then_bb else_bb builder);
        L.builder_at_end context merge_bb

  | A.While (predicate, body) ->
         let pred_bb = L.append_block context "while" the_function in
         ignore (L.build_br pred_bb builder);

         let body_bb = L.append_block context "while_body" the_function in
         add_terminal (stmt (L.builder_at_end context body_bb) body)
```

```
            (L.build_br pred_bb);

          let pred_builder = L.builder_at_end context pred_bb in
          let bool_val = expr pred_builder predicate in

          let merge_bb = L.append_block context "merge" the_function in
          ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
          L.builder_at_end context merge_bb

    | A.For (e1, e2, e3, body) -> stmt builder
          ( A.Block [A.Expr e1 ; A.While (e2, A.Block [body ; A.Expr e3]) ] )
  in

  (* Build the code for each statement in the function *)
  let builder = stmt builder (A.Block fdecl.A.body) in

  (* Add a return if the last block falls off the end *)
  add_terminal builder (L.build_ret (L.const_int (ltype_of_typ fdecl.A.ftyp) 0))
 in

 List.iter build_function_body functions;
 the_module
```

key_input.c by Gabe

```
#include <stdio.h>
#include <sys/select.h>
#include <unistd.h>
#include <termios.h>

#define NB_DISABLE 0
#define NB_ENABLE 1

int kbhit()
{
    struct timeval tv;
    fd_set fds;
    tv.tv_sec = 0;
    tv.tv_usec = 0;
    FD_ZERO(&fds);
    FD_SET(STDIN_FILENO, &fds); //STDIN_FILENO is 0
    select(STDIN_FILENO+1, &fds, NULL, NULL, &tv);
    return FD_ISSET(STDIN_FILENO, &fds);
```

```
}

void nonblock(int state)
{
   struct termios ttystate;

   //get the terminal state
   tcgetattr(STDIN_FILENO, &ttystate);

   if (state==NB_ENABLE)
   {
      //turn off canonical mode
      ttystate.c_lflag &= ~ICANON;
      //minimum of number input read.
      ttystate.c_cc[VMIN] = 1;
      //Disable echo bit
      ttystate.c_lflag &= ~ECHO;
   }
   else if (state==NB_DISABLE)
   {
      //turn on canonical mode
      ttystate.c_lflag |= ICANON;
   }
   //set the terminal attributes.
   tcsetattr(STDIN_FILENO, TCSANOW, &ttystate);
}


char getKey(){

   char c;
   int i=0;

   nonblock(NB_ENABLE);
   while(!i)
   {
      usleep(1);
      i=kbhit();
      //Do this if there is something in the buffer
      if (i!=0)
      {
         c=fgetc(stdin);
         if (c){
```

```
            i=1;
        }
        else{
            i=0;
        }
    }

  }
    nonblock(NB_DISABLE);
    return c;
}
```

print_helper.c by Gabe

```
#include <stdlib.h>

int _clearScreen();

int _clearScreen(){
  system("clear");
  return 0;
}
```

gobble.sh by Gabe

```
#!/bin/bash -e

callDir="$(pwd)"
cd "$(dirname "$0")"

set +e
if [ $# -eq 1 ]
then
   ./goblin.native < $callDir/$1 >a.ll
else
   ./goblin.native $callDir/$1 <$2 >a.ll
fi
set -e

name=$(basename "$1" | cut -d. -f1)
clang -emit-llvm -S -c csrc/goblin_helper.c
clang  a.ll goblin_helper.ll -o $name &> /dev/null
rm a.ll goblin_helper.ll
echo "Successfully Gobbled"
```

```
echo "Generated $name executable"
```