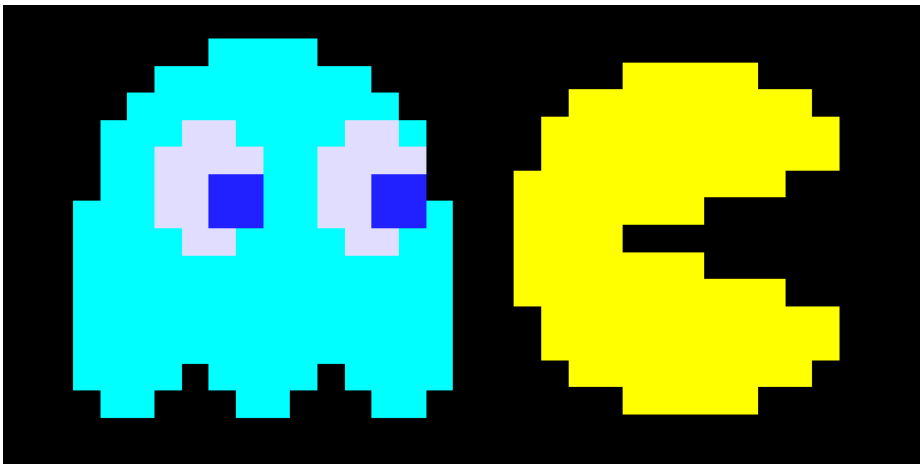


MakerGame Final Report

Cindy Wang (xw2368),
Steven Shao (ys2833),
Yuncheng Jiang (yj2433)

Introduction

Inspired by Game Maker, MakerGame is a C-like language specialized for 2D video games. Many of the primitives in C are present in MakerGame, but the central construct in MakerGame is the object type, similar to a struct, but with added game event functions to specify the behaviors of game objects through their lifetimes. Common game programming patterns like the event loop are thus handled automatically, so that users can focus solely on defining the behaviour of the entities in their game with statements like "create an explosion when I die" or "move left whenever the A key is pressed".

Language Tutorial

Setup

Pre-requisites

- Ocaml/OPAM/LLVM: for compiling the compiler. On Ubuntu 16.04, this might work:

```
sudo apt-get update
sudo apt-get install -y ocaml m4 llvm opam
opam init
opam install llvm.3.8
opam depxt conf-cmake.1
sudo apt-get install llvm-dev
opam depxt conf-llvm.5.0.0
opam depxt conf-pkg-config.1.0
eval 'opam config env’
```

- SFML: for windows, graphics, audio. On Ubuntu 16.04, you can install with the following:

```
sudo apt-get install libsfml-dev
```

Quick Start

Download all the project files, and run the following commands to compile the compiler for MakerGame, along with the runtime library:

```
$ cd MakerGame
$ make
```

If the make succeeds, you should have MakerGame successfully installed, with the key binaries being makergame.native and runtime/libmakergame.a. You can optionally move the archive file for the runtime to your /usr/local/lib, but linking to the runtime somehow or other is necessary.

There is additionally a set of MakerGame library files with utility functions and external declarations from the runtime (like rendering or playing a sound) or from libc (like math functions) in the lib folder. You can move them elsewhere, but in order for the compiler to be aware of them, you need to define the environment variable MAKERGAME_PATH to be the folder containing them. For example, in the MakerGame directory:

```
$ export MAKERGAME_PATH="$(pwd)/lib"
```

Included with the compiler are two games: one of them a simple egg-catching game, and the other one a Yet Another Tetris Clone. To build the egg example (the Tetris is analogous), make sure the compiler's built first. Then run

```
$ cd demo/egg
$ make
$ ./egg.exe
```

Finally, a compile.sh script is provided that takes in a filename (not necessarily in the working directory or the directory of MakerGame) as an argument and builds an executable in the same directory, with minimal setup. It mimicks the behaviour of the Makefiles of the demos.

Crash Course

The following example walks you through implementing a simple egg drop game in which you move a basket around and try to catch the eggs that fall periodically. For this tutorial, make sure the 'res' folder exists when you run the program!

The Basic Framework

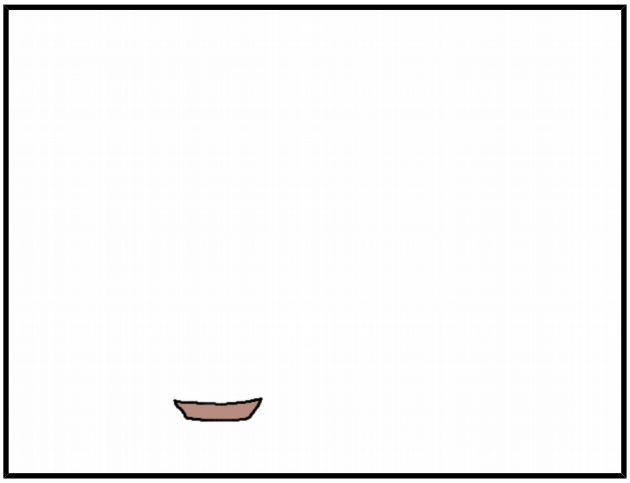
The fundamental building block of a Makergame game is a list of game objects. The special game main object serves as the entry point of the entire game, in the sense that game initialization is precisely the creation of main.

An object's create event handler is called when the object is instantiated. In the following sample code, an object called Player is created by main. The Player object loads an image for its sprite, and every frame, it draws it at a preconfigured position. This functionality is provided by the std::spr namespace, defined in "spr.mg" in the lib folder.

```
// define object type Player
object Player {
  // specify its single member: a sprite
  sprite spr;

  // specify its behaviour when created: load the sprite
  event create {
    // std is a universally accessible namespace, defined in std.mg.
    // it contains helper namespaces like spr and math for common functions.
    spr = std::spr::load(“res/player.png”);
  }
```

```
  // specify its behaviour during the rendering loop: draw its sprite.
  event draw {
    std::spr::draw(spr, 250, 100);
  }
}

// define the behaviour of the entry point
object main {
  event create {
    create Player;
  }
}
```

This code renders the static image at the given location on the screen:



Movement

The step event handler of every instantiated object is run every frame. In this specific example, we continuously check whether the left or right arrow key is pressed, and move the Player basket correspondingly. Similarly to how we used std::spr before, we'll use std::key for keyboard input.

```
object Player {
  sprite spr;
  int x; int y; // define position member variables

  event create {
    spr = std::spr::load("res/player.png");
    x = 350; y = 500; // initialize them here
  }

  event step {
```

```
    // query keyboard input using definitions in std::key
    if (std::key::is_down(std::key::Left)) x -= 5;
    if (std::key::is_down(std::key::Right)) x += 5;
  }

  event draw {
    std::spr::render(spr, x, y); // draw at a variable location
  }
}

object main { event create { create Player; } }
```

More Objects

Let's add a second object and object type to the game for the egg. We want it to fall, so we shift its vertical position every frame. Also, for the sake of this tutorial, let's have it play a sound (using std::snd) when it spawns.

```
sound boinkSound;

object Egg {
  sprite spr;
  int x; int y; float vspeed;
  event create(int my_x, int my_y) {
    x = my_x; y = my_y;
    vspeed = 5;
    spr = std::spr::load("res/egg.png");
    std::snd::play(boinkSound);
  }
  event step { y = y + vspeed; }
  event draw { std::spr::render(spr, x, y); }
}
```

The create event this time takes in a number of arguments, behaving like a constructor in C++ or Java.

In this scenario, by declaring boinkSound outside the definition of any object, we're making it global. The MakerGame runtime reuses sprites from the same image at no extra cost since rendering depends only on the identity of the sprite, not any sort of state like position. In contrast, sounds corresponding to the same file can be reloaded each sound stores state about whether or not it's playing. Here's a simple way to load the sound, make the egg, and set its location.

```
object main {
  event create {
    boinkSound = std::snd::load("res/boink.ogx");
    create Player;
    Egg e = create Egg(400, 0);
  }
}
```

Before moving on, try the program out! You should see the egg fall right through the basket.

Interaction

Let's get the basket to actually catch the egg now. One way to do so is to define a global function that compares Player and Egg positions to find collisions. MakerGame uses the . operator to access member variables (every member is 'public').

```
bool egg_touching_player(Egg e, Player p) {
  // compare the egg centre position to the player bounds
  return (e.x + 16 < p.x + 100 && e.x + 16 > p.x &&
          e.y + 16 < p.y + 10 && e.y + 16 > p.y - 10);
}
```

This can be put anywhere in the file outside of an object definition - function, variable, and object definition order is ignored. The check itself is a little unsightly, but bear with it for now. An alternative would be to define a method on either the Egg or Player, which is functionally equivalent. We'll see how to do this later.

The Player might not know about the egg, but the player can check against every egg alive in the game with a for-each loop in the Player's step event, which iterates over any object type in the game:

```
foreach (Egg e) {
  if (egg_touching_player(e, this)) // this refers to the calling object
    destroy e;
}
```

If you'd like, you can define a variable that keeps track of the number of eggs caught, or some other score-like quantity, and print it out using std::print::i (for integers).

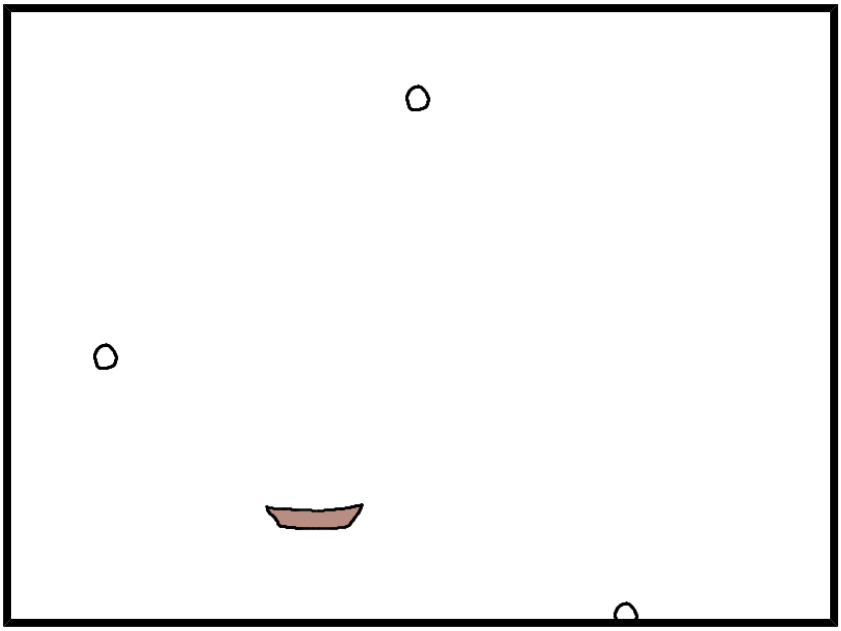
Spawning Eggs

Let's exploit the dynamic object system in MakerGame a little more and spawn eggs periodically. In particular, we can define a Spawner object type that creates an Egg every 50 steps at a random position from the top of the window, using std::math. We can also add a little variance to the speed with the same function.

```
object Spawner {
  int timer;
  int spawn_range;
  event create { timer = 50; spawn_range = 300; }
  event step {
    --timer;
    if (timer == 0) {
      timer = 50;
      spawn_egg();
    }
  }

  void spawn_egg() {
    // std::math::frandom() produces a real from 0 to 1
    int x = 400 + spawn_range * (std::math::frandom() - 0.5);
    Egg e = create Egg(x, 0);
    e.vspeed = 4 + std::math::frandom();
  }
}
```

This time, the Spawner's using its own method spawn_egg, which has access to its member variables, to generate eggs. We can edit the main object to create a Spawner instead of eggs now:

```
object main {
  event create {
    boinkSound = std::snd::load("res/boink.ogx");
    create Player;
    create Spawner;
  }
}
```



Here's a screenshot of the previous sample code.

The Game Object Game Object

You might've noticed the level of redundancy in some of the object definitions, especially between the player and the egg, which both move and draw at its location. std::game (provided in "lib/game.mg") provides an object type that handles this common case of positioning, movement, and at-position rendering. We can use it by having our objects inherit from std::game::obj. Here are the objects updated for the 21st century:

```
object Egg : std::game::obj {
  event create(int x, int y) {
    super(x, y, "res/egg.png"); // call std::game::obj's create event
    vspeed = 5;
    std::snd::play(boinkSound);
  }
}

object Player : std::game::obj {
```

```
  event create { super(400, 500, "res/player.png"); }

  event step {
    if (std::key::is_down(std::key::Left)) x -= 5;
    if (std::key::is_down(std::key::Right)) x += 5;
    foreach (Egg e) {
      if (egg_touching_player(e, this)) destroy e;
    }
  }
}
```

std::game::obj has built-in position members, as well as behaviour for movement and sprite-rendering, so we don't need to define the draw function any more and can additionally just call the parent create to set members up.

An additional experimental feature is its use of hitboxes and sprite offsets/origins, specified through member variables but with supporting functions. We can throw away egg_touching_player if we set the appropriate hitboxes: what we really want in this collision check is whether or not the sprites are roughly overlapping.

One way to do so is to call the inherited member function center_hitbox_prop in the Player and Egg creation events to align their hitboxes with their sprites (or more exactly, a square fraction of their sprite area). We can then replace the Player's collision check with the library function std::game::colliding to do roughly the same calculation. Here is the bleeding-edge Player and Egg.

```
object Player : std::game::obj {
  event create {
    super(400, 500, "res/player.png");
    center_hitbox_prop(0.8, 0.5);
  }

  event step {
    if (std::key::is_down(std::key::Left)) x -= 5;
    if (std::key::is_down(std::key::Right)) x += 5;
    foreach (Egg e) {
      if (std::game::colliding(e, this)) destroy e;
    }
  }
}

object Egg : std::game::obj {
  event create(int x, int y) {
    super(x, y, "res/egg.png");
    vspeed = 5;
    center_hitbox_prop(0.6, 0.6);
    std::snd::play(boinkSound);
  }
}
```

The Full Game

Just letting the basket move and the egg fall is fine, but it lacks the kind of emotional pull games provide through careful narrative, world-building and goal-setting. We bravely take some first steps toward this by punishing the player for failure (sending to a game over screen) and rewarding for success (increasing score). The full code is below, and also in the tutorial demo.

```
int score = 0;
sound boinkSound;

object Spawner {
  int timer;
  int spawn_range;
  event create { timer = 50; spawn_range = 300; }
  event step {
    --timer;
    if (timer == 0) {
      timer = 50;
      spawn_egg();
    }
  }

  void spawn_egg() {
    // std::math::frandom() produces a real from 0 to 1
    int x = 400 + spawn_range * (std::math::frandom() - 0.5);
    Egg e = create Egg(x, 0);
    e.vspeed = 4 + std::math::frandom();
  }
}

object Player : std::game::obj {
  event create {
    super(400, 500, "res/player.png");
    center_hitbox_prop(0.8, 0.5);
  }

  event step {
    if (std::key::is_down(std::key::Left)) x -= 5;
    if (std::key::is_down(std::key::Right)) x += 5;
    foreach (Egg e) {
      if (std::game::colliding(e, this)) {
        destroy e;
        ++score;
        std::print::i(score);
      }
    }
  }
}

object Egg : std::game::obj {
  event create(int x, int y) {
    super(x, y, "res/egg.png");
    center_hitbox_prop(0.6, 0.6);
    vspeed = 5;
    std::snd::play(boinkSound);
```

```
  }

  event step {
    if (y > 600) create game_over;
  }
}

object main {
  event create {
    boinkSound = std::snd::load("res/boink.ogx");
    create Player;
    create Spawner;
  }
}

object game_over { // acts as a "state" or "room" in the game
  event create {
    // trick to remove every other object in the game
    foreach (object o) { if (o != this) destroy o; }
  }
  event step { if (std::key::is_down(std::key::Space)) std::game::end(); }
  event draw { std::spr::render(std::spr::load("res/gameover.png"), 0, 0);}
}
```

Next Steps

Where can you go from here? There's a slightly more fleshed-out version of the egg game in the egg demo, which demonstrates custom modules (for drawing numbers) and inheritance between user objects.

For an actual game, check out the Tetris demo, which shows some ways to avoid having to write std:: all the time, how regular, *non*-std::game::obj objects can be versatile as more abstract game entities, and how std::game::room (also an experimental object) can help with changing global game states. Otherwise, go make some games yourself!

Language Manual

As stated in the intro, MakerGame's tokens syntax roughly line up with C, with object definitions similar to C++ classes.

Lexical Conventions

Comments

Block-comments are introduced with `/*` and terminated with `*/`. They cannot be nested.
Line-comments start with a `//` and continue to the end of the line.

Identifiers

Identifiers start with an alphabetic character, followed by any number of alphanumerics or underscores. The following keywords cannot be used as identifiers:

`create`	`destroy`	`draw`	`step`	`delete`
`if`	`else`	`for`	`while`	`foreach`
`break`	`return`	`open`	`true`	`false`
`int`	`bool`	`char`	`float`	`void`
`object`	`sprite`	`sound`	`none`	`event`
`namespace`	`private`	`public`	`extern`	`using`

Literals

1. **Integer** literals are nonempty sequences of digits from 0 to 9.
2. **Floating point** literals consist of a decimal point with a non-empty sequence of digits on at least one side of it.
3. **Boolean** literals are either `true` or `false`.
4. **String** literals are any sequence of ASCII characters (minus double quotes and backslashes) surrounded by double quotes. In a literal, \" represents a double quote and \\ represents a backslash.
5. **Array** literals/initializers are sequences of comma-separated expressions surrounded by square brackets.
6. The single **object** literal is the keyword `none`.

Operators

In order of decreasing precedence,

operator	**associativity**	**description**
`()`	left-to-right	function call
`::`	left-to-right	namespace access
`. []`	left-to-right	object member access, array subscripting
`++ --`	left-to-right	prefix increment and decrement
`! - (unary)`	right-to-left	logical and numerical negation
`* / %`	left-to-right	multiplication, division, modulo
`+ - (comp)`	left-to-right	addition, subtraction
`== <= >= < >`	left-to-right	boolean relations
`&& ||`	left-to-right	Logical AND, logical OR (no short-circuit)
`= += -= *= /=`	right-to-left	assignment, operator assignment

Misc. punctuation

Some punctuation also act as separators or parts of more complex structures.

1. **Parentheses** group expressions in order to force precedence order.
2. **Commas** separate function arguments and elements in an array definition.
3. **Semicolons** terminate statements.
4. **Curly Braces** enclose concrete namespaces, code blocks (statements and definitions) and object definitions.
5. **Square Brackets** enclose array definitions.
6. **Whitespace** separates tokens.

Datatypes

Primitives

Following is a list of data types and sample declarations (*`type ident = literal`*).

1. **int**: 32-bit signed integer value
   `int x = 5;`
2. **float**: double-precision floating point value
   `float y = 3.14;`
3. **bool**: one of ‘true’ or ‘false’
   `bool b = true;`
4. **string**: an opaque reference to an array of bytes representing a string.
   `string s = "hello.mg";`
5. **sprite**: an opaque handle to an image that can be drawn to the screen.
   `sprite s = spr::load("player.png");`
6. **sound**: an opaque handle to a sound file that can be played.
   `sound s = snd::load("level.ogg");`

Fixed-length arrays

Arrays are a contiguous sequence of objects of a particular type. They can be instantiated with

`typename identifier``[`*`size`*`]`

Array subscripting can be used to access elements or specific dimensions of the array, and they can be passed into and returned from functions like any other type. Examples:

```
bool answers[3] = [false, true, false];
int mat[2][3] = [[1, 2, 3], [4, 5, 6]];
int row = mat[1]; // [4, 5, 6]
```

Game objects

Game objects are a class of types consisting of a collection of primitives, arrays, and other game objects, combined with four event handlers with fixed names that define the object's behaviour in the game. Game object types can inherit from other game object types to form a hierarchy (singular inheritance with no cycles). Their details are given below in the Game Objects section.

Type conversions

A value of a certain type may be implicitly converted to one of a different type in certain scenarios:

1. assignment
2. array literal type regularization
3. comparison
4. function call arguments

In these cases, basic type conversion rules apply:

1. A child object type may be converted to a parent type. (`none` is a child of all game objects)
2. An int may be converted to a float.

3. A float may be converted to an int.

Some scenarios do not require necessarily a single value be converted to another type, but simply to match the types of multiple values. In this case, only an attempt is made for each value to convert the other values into that type. So, it is not possible to compare instances of two different game objects even though they are both convertible to the same type (they both inherit from `object`).

Lvalue References

Only certain types of values are legal in some of the operators that modify one of their operands (assignment, increment/decrement, and operator assignment). These lvalues take exactly one of the following forms:

1. An identifier
2. A member variable (*identifier* . *name*)
3. An element of an array that is an lvalue
4. The result of an assignment (refers to the same as the LH expression of that assignment)
5. The result of an increment/decrement (analogous)
6. The result of an operator assignment (analogous)

Expressions

Expressions evaluate to a value, with a possible side effect. The following are valid expressions:

Literals

Defined above, literals directly specify the value of a type in code.

Identifiers

Identifiers refer to a particular variable in either local, object, or global scope. The identifier can include a prefix to indicate the namespace in which the identifier is defined, so that the following are valid identifiers:

name
name1 `::` *name2* `::` `...` `::` *namen* `::` *name*

Where the latter refers to a variable in the nested namespace *name1* :: ... :: *namen*.

Operator Results

These include the results of the following operations:

1. Binary arithmetic (e.g., *expr1* + *expr2*, or *expr1* || *expr2*)
2. Unary operations (e.g., -*expr*, or --*expr*)
3. Assignments (*ident* = 3 evaluates to 3)
4. Operator assignments (*ident* +=3 evaluates to the new value of *ident*)

Object Commands

```
create identifier
destroy identifier
delete identifier
```

are all valid expressions, a special case of a unary operator result. The first evaluates to the created object (and the *identifier* is the possibly-namespace-prefixed object type), while the last two to *void* (and the *identifier* is refers to the object reference).

Member access

A special case of a binary operator result, this type of expression evaluates to the value of the member of the given object. The syntax is as follows:

```
expr . name
```

Where the expression evaluates to the object from which to get member `name`.

Function Calls

Function calls evaluate to the return value of the function, and follow the syntax:

```
identifier ( expr1, expr2, ...)
```

As with identifiers, function call names can include a namespace access prefix. Method calls are also valid expressions, but follow this syntax:

```
expr1 . name ( expr2, expr3, ...)
```

where the first expression refers to the calling object, and the name the method to call.

Statements and control flow

Function and object event-handler bodies are code-blocks, which contain a list of statements. Like in C, statements are either:

1. expressions
2. brace-enclosed code blocks
3. declarations (`typ identifier [= expr];`), which add new symbols to the scope, up until the end of the code block.
4. `return` statements, which exit a function and assign its value
5. control flow structures

Following is a sample of these flows:

Conditionals

As in C, a conditional statement executes the branch corresponding to the first satisfied predicate. The expressions in the condition must resolve to type `bool`.

```
if (expr1) statement
else if (expr2) statement
else statement
```

In this case and the cases below, any variables introduced by the statements in the structure are not held over past the end of the structure.

Basic Loops

As in C, a while loop repeatedly executes its body as long as the boolean expression is true.

```
while (bool) statement
```

A for loop is functionally equivalent to a block containing a while loop with a statement (*stmt1*) before its body and a statement at the end of each iteration of its body (*stmt2*).

```
for (stmt1; pred; stmt2) statement
```

A `break` statement can be used to exit the innermost loop it is contained in.

Object Iteration

A foreach loop iterates through every object of a particular game object type (*objtype*).

```
foreach (objtype identifier) statement
```

foreach loops can also be used to iterate through any game object using the objtype `object`. This can be used, for example, to remove every object but a particular one from the game (e.g., in the case of switching scenes):

```
foreach (object o) { if (o != this) delete o; }
```

Functions

One can define a function in any namespace with the following syntax:

```
returntype functionname(type1 arg1, type2 arg2, …) { statements }
```

Following the semantics of C, when the function must return a value of type *returntype*, and calls to the function must match in argument count and type (with comment above about conversion) to the function specification. The function body is a code block with additional access to function arguments and any outside identifier accessible from the namespace, e.g. from inner namespaces.

External Functions

Functions defined in code outside of MakerGame can be referred to with the following syntax:

```
extern returntype functionname(type1 arg1, type2 arg2, ...);
```

During compilation, the linker will verify that external functions are defined in one of the linked libraries.

Game Objects

Definition

Their generic definition is as follows:

```
object objectname : parentname { // parent optional
      // variable declarations
      type1 name1; type2 name2; // etc.

      ftype1 fname1(...) { ... }
      // etc. cannot be external

      // all definitions are optional, with defaults calling parent
      // create can have parameters; not the others.
      event create(type1 arg1, type2 arg2, ...) { ... }
      event destroy { ... }
      event step { ... }
      event draw { ... }
}
```

Behaviour

These objects can be instantiated, and every frame, the body of the *step* handler of every instantiated object is run, followed by the body of every *draw* handler for rendering. An object's *create* and *destroy* event handlers are called when the object is instantiated and removed from the game, respectively.

Method & Event Bodies

Bodies of event handlers and methods have access to member variables, locally declared variables, and global variables. They also have access to global functions and object methods. One can also refer to the calling object itself with `this`.

If an object type has a parent, all member variables and methods are inherited from the parent. Additionally, `super()` is available in event handlers to call the same event handler of the parent.

Manipulation

References

If an object type is defined, references to that object type can be declared using just the name of the object type. One can declare a reference to any game object with the type `object`. For example:

```
object Player {
  int x;
  void attack();
}

void compute() {
  object o;
```

```
  Player p;
}
```

Operations and Assignment

To instantiate a Player, assign it to a reference, and destroy it, one can use the following syntax:

```
Player p = create Player;
destroy p;
```

The `delete` keyword is also provided to remove an object from the game without activating its destroy event handler.

The `none` object also be assigned to any reference to indicate pointing to no object.

An object of a particular type can be assigned to a reference to any of its parent types (so any object can be assigned to a reference to `object`), but parent handlers can only access methods and variables available from the parent. Only object events are virtual, so if a child defines a method of the same name and signature as a parent, a reference to the parent object will result in the parent method being called.

Comparison

Objects may be compared with one another with the == operator. `none == x` if and only if `x` points to the singleton `none`.

Access

Member access and method calls are analogous to in C++, using the . operator:

```
Player p = create Player;
p.x = 3;
p.attack();
```

Namespaces

To encourage modularity and code reuse, MakerGame supports namespaces that define boundaries between units of code and allow for access across files. There are three types of namespaces:

Concrete

A concrete namespace is a namespace with its body definition included with its declaration. The generic syntax is as follows:

```
namespace ns { ... }
```

Where the structure of the namespace interior is the same as that of the program, defined below. Access to entities in the namespace is done with the :: operator, however, an inner namespace has no access to anything declared outside of it, for dependency reasons. For example,

```
namespace ns { int x; }
```

```
int compute() { ns::x = 3; }
```

Alias

An alias simply points to another namespace, and is typically used as shorthand for inner namespaces of other namespaces.

```
namespace alias = ns;
```

Anything accessed through an alias refers to the same thing accessed through the original namespace. For example:

```
namespace outer { namespace inner { int x; } };
namespace alias = outer::inner;
int compute() { alias::x = 3; return outer::inner::x; } // returns 3
```

Files

A file namespace functions the same as a concrete namespace, except its contents are the contents of the file it refers to.

```
namespace file = open "file.mg";
```

References from different namespaces referring to the same file (i.e., same UNIX realpath) resolve to the same entity. Circular inclusion of files is forbidden, so that more generally, namespace access restrictions form a DAG.

Files are searched for first in the directory of the source calling the open file (for multi-file projects), and then the directory defined by the environment variable `MAKERGAME_PATH` (for standard library access).

Using

If a namespace is accessible from an outer namespace, the keyword `using` can be used to make accessible the functions and variables defined in the outer namespace. For example,

```
namespace ns = { int x; }
using ns;
int compute() { return x; }
```

It is worth noting that top-level declarations in a namespace can be put in any order, so that even if a using declaration is at the end of a namespace body, the identifiers it puts into scope are accessible in any function or method defined in the namespace.

Access Modifiers

Namespaces can additionally be marked `private` so that any namespace other than the enclosing namespace has no access to the definitions inside the private namespace. For example,

```
namespace ns = {
  private namespace p { int x; }
```

```
  void incr() { ++x; } // legal
}
int compute() { ++ns::p::x; } // illegal
```

Program Structure

A program consists of a sequence of declarations of the following types:

1. Variable declarations
2. Function declarations
3. Namespace and using declarations
4. Game object declarations

The order of these declarations has no effect on scope within the namespace, so that the effects of any declaration is visible by any other declaration in that namespace.

Namespace variable declarations can also optionally include an assignment, but only to constant expressions, which are defined as either non-array literals, or array literals of constant expressions.

While there may be multiple files, the single 'seed' file given to the compiler must have a game object of type `main` defined. The resultant program's entry point simply creates an instance of `main`.

Reflection

Compared to our initial LRM submission, the one we included here with our final report depends less on the specifics of the underlying multimedia library. In particular, there is no prespecified list of built-in functions, but instead an option to declare external functions and to link files to each other is provided to enable customizable libraries and modules. Thus the only hard-coded option in MakerGame is the inclusion of the standard library from std.mg in every namespace. We discuss later in the **Runtime Environment** where the remaining game-specific elements come into play.

To see the **grammar**, please refer to the AST, scanner, and parser files in the appendix.

Project Plan

Team Member Roles and Responsibilities

Role	Responsibility
Manager	Yuncheng Jiang
Language Guru	Steven Shao
System Architect	
Tester	Cindy Wang

All the team members are responsible for the role as assigned, but we also embraced changes during the development. We all discussed on the design of the language. Since we only have three of us, we merged some responsibilities as well during the project.

Programming Style Guide

Style guidelines for OCaml code roughly followed these rules:

1. 90 character limit, most of the time
2. Descriptive names for each variable
3. Use ocp-indent
4. Single-line let definitions are permissible, but multi-line let definitions should have the definition on its own line, with the in keyword on the following line.

Formatting for MakerGame code resembled LLVM coding style rules for clang-format.

Collaboration-wise, we adhered to these guidelines:

1. Create a branch for each feature, and a PR for merging to master
2. Write multiple passing and failing tests for each feature

Project Timeline

Week	Task	Milestone
Sep 17 - Sep 23	Come up with general idea about MakerGame	
Sep 24 - Sep 30	Added example program for MakerGame	Sep 26 Proposal
Oct 1 - Oct 7	Discuss the feature included	
Oct 8 - Oct 14	Go through Micro C	
Oct 15 - Oct 21	Decide the syntax for MakerGame	Oct 16th LRM
Oct 22 - Oct 28	Minutiae: strings, printing, generalized statements	
Oct 29 - Nov 4	Linking to the external library Game object structures	
Nov 5 - Nov 11	Game object semantics and loops Demo: Egg Drop	Nov 8 Hello World
Nov 12 - Nov 18	Foreach, Namespaces and files, miscellany	
Nov 19 - Nov 25	Namespaces and files, object methods	THANKSGIVING
Nov 26 - Dec 2	Namespaces and files, object methods, miscellany	
Dec 3 - Dec 9	Namespaces and files, object inheritance	
Dec 10 - Dec 16	Namespaces and files, object inheritance Demo: Tetris	
Dec 17 - Dec 23	Wrap up the final report Prepare final presentation	Dec 20 Presentation

Planning Process

We held a weekly meeting schedule. At the start of the project, most meetings were about the high level design of the language, and we consulted with the professor and multiple TAs for feedback before proposal submission. The specification of our language largely tried to match a kind of happy medium between GameMaker's features and C/C++'s expressivity. After the LRM, our meetings were mostly pair programming and discussion on different individual features. For some of the weeks, we had two meetings every week: one with our TA to ask questions and another one among ourselves to implement features. After the Hello World milestone, each of us were responsible for more individual add-on features.

We used a Github repository to keep track of milestones/goals, as well as progress for each feature.

Development Process

At the start of the project, we worked together to decide the high-level design of the language. Then we went through microC together to understand the structure and details of the compiler, before adding features for linking to external libraries and setting up the runtime structure and iteration flow to produce a working MVP by the Hello World deadline.

After the deadline, work split into a few large remaining milestones, and then some incremental language features (labelled miscellany in the timeline), like

1. Implicit type conversions
2. Additional basic types like arrays
3. Improved scopes (e.g., member access without the *this* keyword)
4. Additional operators and expanded variable declaration syntax

The larger milestones included

1. File and namespace support
2. Decoupling from the external runtime by minimizing the number of built-ins
3. Object methods and inheritance
4. Generalizing the game loop to a foreach loop

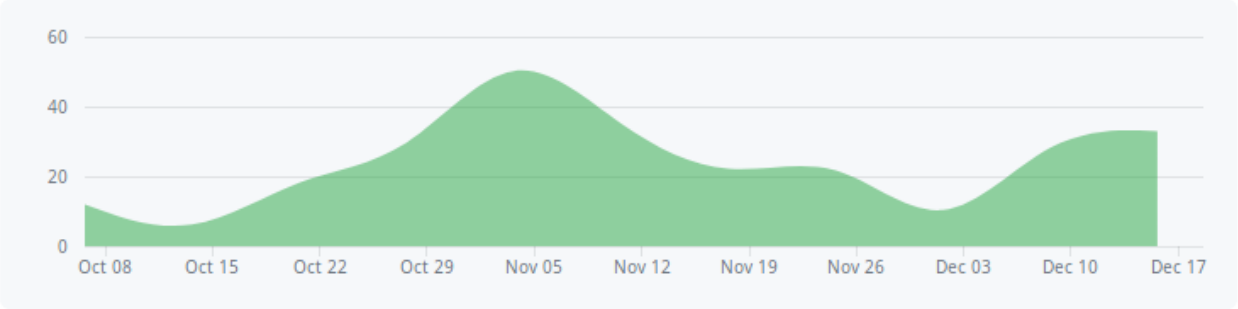
We met weekly to catch up with each other and do pair programming, but additionally, some of us spent a lot of extra time outside of meetings to take on and implement new features.

Environment

- Github: for version control, project management, and synchronization
- OCaml 4.02.3, LLVM 5.0.0, SFML 2.4
- Various operating systems (Bash on Windows, OSX, Arch Linux)
- Google docs/slides for documents and presentations.
- Spacemacs and Visual Studio Code for writing code.

Project Log

Here's the contribution chart on our github:



Total: 16786 insertions, 9651 deletions

In the list below, we did a fair amount of pair programming, so not all commits are necessarily attributed to the correct author.

```
Author: Steven <ys2833@columbia.edu>
Date:   Wed Dec 20 02:40:45 2017 -0500
another array test

Author: Steven <ys2833@columbia.edu>
Date:   Tue Dec 19 23:13:51 2017 -0500
tetris working

Author: Steven <ys2833@columbia.edu>
Date:   Tue Dec 19 21:18:57 2017 -0500
add difficulty level, logo

Author: Steven <ys2833@columbia.edu>
Date:   Tue Dec 19 05:28:25 2017 -0500
modulo times divide precedence

Author: Steven <ys2833@columbia.edu>
Date:   Tue Dec 19 04:54:49 2017 -0500
compile script, tutorial

Author: Steven <ys2833@columbia.edu>
Date:   Tue Dec 19 03:13:26 2017 -0500
compile.sh

Author: Steven <ys2833@columbia.edu>
Date:   Tue Dec 19 00:40:08 2017 -0500
nice score

Author: Steven <ys2833@columbia.edu>
Date:   Tue Dec 19 00:27:18 2017 -0500
pretty done tetris

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 22:57:40 2017 -0500
sad... check in list remove

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 22:57:25 2017 -0500
next piece, tests
```

```
Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 21:34:14 2017 -0500
better tetris

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 20:09:19 2017 -0500
clean up demo

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 20:06:31 2017 -0500
zero init

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 19:09:17 2017 -0500
head and tail working really

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 18:40:54 2017 -0500
number display

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 16:06:25 2017 -0500
bug fix: formal types in inner namespace now are recognized

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 15:36:26 2017 -0500
game module

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 15:34:49 2017 -0500
addasn between floats and ints

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 15:08:37 2017 -0500
terser egg

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 15:08:11 2017 -0500
link to libc, improve math

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 14:45:11 2017 -0500
modulo operator

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 14:07:36 2017 -0500
fix string_ofs in ast

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 13:46:00 2017 -0500
using, without private

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 05:45:36 2017 -0500
modularized library
```

```
Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 05:16:57 2017 -0500
updated demos a bit

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 04:47:25 2017 -0500
added delete keyword

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 04:33:13 2017 -0500
introduce delete into vtables so destroy can be a regular event. no more
postprocessing

Author: Steven <ys2833@columbia.edu>
Date:   Mon Dec 18 03:39:56 2017 -0500
archive file

Author: Steven <ys2833@columbia.edu>
Date:   Sun Dec 17 03:51:47 2017 -0500
better tetris

Author: Steven <ys2833@columbia.edu>
Date:   Sun Dec 17 03:49:17 2017 -0500
runtime fixes: setting window title/size

Author: Steven <ys2833@columbia.edu>
Date:   Sun Dec 17 02:20:48 2017 -0500
moved lib to runtime

Author: Steven <ys2833@columbia.edu>
Date:   Sun Dec 17 01:49:30 2017 -0500
super on step and draw

Author: Steven <ys2833@columbia.edu>
Date:   Sun Dec 17 01:19:21 2017 -0500
format, inheritance chain consolidation and correctness

Author: Steven <ys2833@columbia.edu>
Date:   Sat Dec 16 23:03:42 2017 -0500
correct virtual events

Author: Steven <ys2833@columbia.edu>
Date:   Sat Dec 16 17:45:24 2017 -0500
tests without core

Author: Steven <ys2833@columbia.edu>
Date:   Sat Dec 16 17:43:11 2017 -0500
remove dependency of core, use own c stub

Author: Steven <ys2833@columbia.edu>
Date:   Sat Dec 16 16:19:33 2017 -0500
window functions

Author: Steven <ys2833@columbia.edu>
```

```
Date:   Sat Dec 16 15:48:05 2017 -0500
better egg demo, todo

Author: Steven <ys2833@columbia.edu>
Date:   Sat Dec 16 01:04:05 2017 -0500
updated demos to use constants

Author: Steven <ys2833@columbia.edu>
Date:   Sat Dec 16 00:58:25 2017 -0500
constexprs to globals - non-string literals and arrays of such

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 22:45:25 2017 -0500
is_alive query

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 22:40:07 2017 -0500
allow decl object except in globals

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 22:19:45 2017 -0500
foreach over objects

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 22:06:57 2017 -0500
none object

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 21:48:08 2017 -0500
object comparisons

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 21:43:25 2017 -0500
fix compare bug

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 21:05:05 2017 -0500
test harness improvement: step limit

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 20:23:02 2017 -0500
everything inherits from object

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 20:03:19 2017 -0500
slight cleanup/consolidation of getting gameobj_decl in codegen

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 18:41:39 2017 -0500
updated demo. added some tetris. obj_end in codegen fixed. std lib a little
better

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 05:03:19 2017 -0500
updated demos
```

```
Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 04:57:36 2017 -0500
resolve files: absolute path, relative to current file, then system path.
semant converts all relative paths to absolute so codegen has simpler use.
tests updated. std.mg moved to lib folder, which is put in path

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 03:12:08 2017 -0500
error test format changed after using Core

Author: Steven <ys2833@columbia.edu>
Date:   Fri Dec 15 03:11:35 2017 -0500
using core.filename

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 18:38:45 2017 -0500
conversion from child to parent

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 18:25:20 2017 -0500
tests

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 18:25:13 2017 -0500
expose super() in object create

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 17:37:43 2017 -0500
remove nodes and call parent destroy in destroy event

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 16:44:03 2017 -0500
more tests

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 16:13:36 2017 -0500
slightly better tests, cleaner codegen

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 16:08:56 2017 -0500
more tests

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 16:05:01 2017 -0500
some tests

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 16:04:53 2017 -0500
set gameobj body after defining llns. include parent scopes. link/unlink
parent nodes in create/destroy.

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 15:58:35 2017 -0500
```

```
semant: include parent scope in member/membercalls/defns of child fns. super
keyword basics

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 15:23:24 2017 -0500
ast, parser, scanner for inheritance

Author: Steven <ys2833@columbia.edu>
Date:   Thu Dec 14 02:48:40 2017 -0500
BUG FIXES! global scope overriding obj scope, declaring nonexistent objs

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 28 02:17:16 2017 -0500
node as second elem in gameobj_t struct

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 28 01:29:29 2017 -0500
vtable for events

Author: Steven <ys2833@columbia.edu>
Date:   Wed Dec 13 20:10:59 2017 -0500
more tests

Author: Yuncheng <yj2433@columbia.edu>
Date:   Thu Nov 30 22:05:51 2017 -0500
allow one line definition in for loop

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 30 23:41:20 2017 -0500
todos

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 30 22:12:30 2017 -0500
cleanup

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 30 21:50:05 2017 -0500
array literals

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 30 21:48:52 2017 -0500
array index/size order correct

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 28 22:01:40 2017 -0500
more array tests

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 28 21:46:01 2017 -0500
vdef id name consistency

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 28 21:42:00 2017 -0500
more array tests
```

```
Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 28 21:41:08 2017 -0500
better syntax for arrays

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 28 21:24:58 2017 -0500
preliminary arrays; bad subscript syntax

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 30 23:07:10 2017 -0500
indentation

Author: Cindy <xw2368@columbia.edu>
Date:   Thu Nov 30 22:48:47 2017 -0500
Int Float conversions

Author: Cindy <xw2368@columbia.edu>
Date:   Thu Nov 30 22:08:48 2017 -0500
Codegen considers Conv

Author: Cindy <xw2368@columbia.edu>
Date:   Thu Nov 30 21:38:28 2017 -0500
check_assign return type, new value (either original or Conv) tuple

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 28 21:27:57 2017 -0500
slight cleanup

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 28 00:30:25 2017 -0500
correct namespace loop resolution

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 27 22:47:48 2017 -0500
better scope test

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 27 22:13:35 2017 -0500
fixed bug: add_to_scope overrode higher priority with lower priority

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 27 22:13:01 2017 -0500
contructor with arguments

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 27 20:57:10 2017 -0500
gameobj events as list of functions

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 27 22:29:24 2017 -0500
cleanup: executables, indent

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 27 22:28:25 2017 -0500
semant stmt should return revised stmt
```

```
Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 27 22:25:13 2017 -0500
cleanup

Author: Yuncheng <yj2433@columbia.edu>
Date:   Mon Nov 27 21:57:31 2017 -0500
add test cases for one-line decl

Author: Yuncheng <yj2433@columbia.edu>
Date:   Mon Nov 27 21:40:30 2017 -0500
Merge from master

Author: Yuncheng <yj2433@columbia.edu>
Date:   Mon Nov 27 21:07:04 2017 -0500
one line definiton

Author: Steven <ys2833@columbia.edu>
Date:   Sat Nov 25 03:24:43 2017 -0500
test for private ns through public alias

Author: Cindy <xw2368@columbia.edu>
Date:   Fri Nov 24 19:48:42 2017 -0500
Minor changes

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 24 18:50:27 2017 -0500
update inc/dec tests to load-file

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 24 18:49:55 2017 -0500
proper ns checking: aliases after concretes

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 24 16:48:24 2017 -0500
private namespaces

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 24 13:20:31 2017 -0500
disallow void main() as entry; rewrite tests

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 24 03:24:56 2017 -0500
moved all preloaded functions to std namespace

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 24 02:33:54 2017 -0500
tiny codegen cleanup; check for alias to nonexistent

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 24 02:28:52 2017 -0500
load files, relative to compiler directory

Author: Cindy <xw2368@columbia.edu>
Date:   Fri Nov 24 17:07:15 2017 -0500
```

```
Modified tests

Author: Cindy <xw2368@columbia.edu>
Date:   Fri Nov 24 16:53:57 2017 -0500
Updated .err files

Author: Cindy <xw2368@columbia.edu>
Date:   Fri Nov 24 16:45:42 2017 -0500
Increment and Decrement

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 24 15:06:52 2017 -0500
check lvalue in asnop

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 24 02:01:15 2017 -0500
split declaring and defining llns contents. moved fn_decls to slightly outer
scope

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 21 03:23:40 2017 -0500
microc -> makergame

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 21 03:16:21 2017 -0500
allow alias decls anywhere, but only forbid circular refs

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 21 02:44:01 2017 -0500
represent ns alias as id list instead of chain

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 21 02:39:36 2017 -0500
namespace aliases

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 17 13:17:56 2017 -0500
cleanup ast

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 17 13:13:53 2017 -0500
cleanup semant; test for bool comp

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 17 13:03:20 2017 -0500
clean a few TODOs

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 17 12:52:19 2017 -0500
cleanup: modules for func/var, minusasn to subasn, expr arm mvt

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 17 03:12:09 2017 -0500
prevent duplicate member fns
```

```
Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 17 03:09:24 2017 -0500
no duplicate namespaces

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 17 02:45:05 2017 -0500
tests for namespaces

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 17 02:44:54 2017 -0500
llfn naming, references to objtypes inside namespaces

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 17 02:10:02 2017 -0500
first try: relevant exprs keep a namespace chain. obj types also have
relative chain

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 18:52:08 2017 -0500
symbols mangled w namespaces

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 12:17:03 2017 -0500
node ends in llgameobj

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 04:12:38 2017 -0500
use Llast.namespace



Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 03:54:24 2017 -0500
updated Llast.gameobj to be record

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 03:44:48 2017 -0500
LLAST + use in gameobjs for codegen

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 02:54:51 2017 -0500
moved lots of fns around in codegen to reduce dependencies

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 02:31:49 2017 -0500
semant error reports with namespace

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 02:15:01 2017 -0500
recursively check namespaces

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 02:14:42 2017 -0500
move helper checks in semant outside of check
```

```
Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 02:05:19 2017 -0500
ast for declaring namespace

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 01:43:07 2017 -0500
namespace structure in ast

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 01:18:06 2017 -0500
gameobjs into bind

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 00:59:24 2017 -0500
flipped bind definition

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 16 00:52:35 2017 -0500
function as a bind

Author: Yuncheng <yj2433@columbia.edu>
Date:   Tue Nov 14 20:46:59 2017 -0500
implement += -= *= /= operators

Author: Yuncheng <yj2433@columbia.edu>
Date:   Mon Nov 13 10:04:28 2017 -0500
add test case for += operator

Author: Yuncheng <yj2433@columbia.edu>
Date:   Sun Nov 12 19:25:56 2017 -0500
add += as more operators

Author: Yuncheng <yj2433@columbia.edu>
Date:   Sun Nov 12 19:24:48 2017 -0500
add += as more operators

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 12 14:52:49 2017 -0500
extract check_lvalue in semant

Author: Yuncheng <yj2433@columbia.edu>
Date:   Sun Nov 12 14:28:49 2017 -0500
update the egg.mg based on parser

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 12 02:49:11 2017 -0500
propagate function scope, record if function is member or global, member
functions in event/object scope

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 12 01:55:26 2017 -0500
member calls through .

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 12 03:36:46 2017 -0500
```

```
small definition move

Author: Steven <ys2833@columbia.edu>
Date:   Sat Nov 11 03:27:30 2017 -0500
build_function shorthand

Author: Steven <ys2833@columbia.edu>
Date:   Sat Nov 11 03:18:31 2017 -0500
generation of object methods... presumably

Author: Steven <ys2833@columbia.edu>
Date:   Sat Nov 11 02:53:07 2017 -0500
move fn and event decls to better spot in codegen

Author: Steven <ys2833@columbia.edu>
Date:   Sat Nov 11 02:05:11 2017 -0500
member functions up until semant

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 12 03:14:01 2017 -0500
also have assignment be lvalue

Author: Steven <ys2833@columbia.edu>
Date:   Sat Nov 11 23:24:25 2017 -0500
generalized member access with PERIOD operator

Author: Steven <ys2833@columbia.edu>
Date:   Sat Nov 11 02:02:11 2017 -0500
rename some function llvalues to be unique

Author: Steven <ys2833@columbia.edu>
Date:   Sat Nov 11 00:29:25 2017 -0500
keywords for defining objects and events

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 10 23:53:06 2017 -0500
member variables without 'this'

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 10 22:57:03 2017 -0500
indent

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 10 22:27:34 2017 -0500
a few more tests

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 10 21:41:06 2017 -0500
removed printbig, added test lib, fixed binop for bools

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 10 14:17:25 2017 -0500
break tests

Author: Steven <ys2833@columbia.edu>
```

```
Date:   Fri Nov 10 14:15:30 2017 -0500
break out of loops

Author: Cindy <xw2368@columbia.edu>
Date:   Fri Nov 10 18:18:20 2017 -0500
Fixed conflicts

Author: Cindy <xw2368@columbia.edu>
Date:   Fri Nov 10 18:07:01 2017 -0500
Neater code for codegen binop functions

Author: Cindy <xw2368@columbia.edu>
Date:   Fri Nov 10 00:24:53 2017 -0500
Enable floating points operations

Author: Cindy <xw2368@columbia.edu>
Date:   Fri Nov 10 00:24:53 2017 -0500
Enable floating points operations

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 10 12:12:32 2017 -0500
disallow shadowing in same block

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 9 04:01:11 2017 -0500
some lib functions, demo

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 9 03:24:35 2017 -0500
fancy file extension

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 9 03:08:34 2017 -0500
external sounds

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 9 02:22:47 2017 -0500
moved stmt out of build_block; removed build_block

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 9 01:59:46 2017 -0500
tail node padding to linked lists for correct behaviour when creating during
step/iter

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 9 01:31:35 2017 -0500
global_event using build_if

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 9 01:14:53 2017 -0500
build_node_loop and build_object_loop using build_while and build_if

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 23:51:37 2017 -0500
```

```
extract build_if build_while

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 21:45:16 2017 -0500
tests

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 21:20:29 2017 -0500
lazy destroy

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 19:04:54 2017 -0500
current + next ptrs in node loop

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 02:20:15 2017 -0500
foreach

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 02:19:49 2017 -0500
fixed bug in linked list traversal (used to be backwards)

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 01:55:59 2017 -0500
fixup for object and node loop

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 01:40:02 2017 -0500
build_object_loop for particular objects

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 00:40:38 2017 -0500
tests for create-destroy

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 00:25:16 2017 -0500
move code to build_object_loop

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 03:52:22 2017 -0500
this identifier done right

Author: Steven <ys2833@columbia.edu>
Date:   Wed Nov 8 00:24:04 2017 -0500
restrictions on and tests for 'this' keyword

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 7 23:20:18 2017 -0500
```

```
unlink obj-particular nodes on destroy; store type info in destroy operator

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 7 01:59:00 2017 -0500
format, todo

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 7 03:17:43 2017 -0500
rename symbols to scope in semant

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 7 03:15:39 2017 -0500
cleanup: lookup and expr out of block fn in semant and codegen

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 7 03:06:21 2017 -0500
stuff can go after a return

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 7 02:34:03 2017 -0500
declarations interwoven with statements

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 6 17:31:13 2017 -0500
code blocks can have declarations

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 6 17:26:59 2017 -0500
no more duplicating format strings

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 6 16:59:21 2017 -0500
move a few functions around

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 7 12:41:57 2017 -0500
comment fix

Author: Steven <ys2833@columbia.edu>
Date:   Tue Nov 7 12:35:16 2017 -0500
add todos

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 6 02:03:32 2017 -0500
particular object node field

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 6 01:38:33 2017 -0500
```

```
extra linked list field in particular obj struct

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 6 01:13:04 2017 -0500
inline linked lists

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 6 01:12:18 2017 -0500
tags debug

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 5 19:49:57 2017 -0500
objref with ids

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 5 18:38:43 2017 -0500
obj id increment

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 5 18:36:11 2017 -0500
destroy event fix

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 5 18:32:21 2017 -0500
extra id field

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 5 03:05:31 2017 -0500
create and destroy in codegen, semant. member access

Author: Steven <ys2833@columbia.edu>
Date:   Sun Nov 5 03:04:38 2017 -0500
small lib changes, comment

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 3 16:32:50 2017 -0400
playing around with construction

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 3 16:32:39 2017 -0400
node type (untested)

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 3 16:31:44 2017 -0400
syntax and semant - type constructor conflicts tho

Author: Cindy <xw2368@columbia.edu>
Date:   Mon Nov 6 18:42:03 2017 -0500
```

```
Semant add typed operators

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 6 15:51:54 2017 -0500
find_opt -> mem

Author: Steven <ys2833@columbia.edu>
Date:   Mon Nov 6 15:47:12 2017 -0500
union -> merge

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 3 16:55:26 2017 -0400
gameobj ast submodule

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 3 02:30:00 2017 -0400
semant converts main fn to main gameobj. tests work

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 3 02:29:22 2017 -0400
small fixes to lib and semant

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 3 01:26:38 2017 -0400
parse lists left-to-right where possible

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 3 00:36:59 2017 -0400
object member variable access semantics

Author: Steven <ys2833@columbia.edu>
Date:   Fri Nov 3 00:07:56 2017 -0400
syntax for declaring obj vars and getting members

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 2 23:47:47 2017 -0400
TODOs, named gameobj struct LLVM defns

Author: Steven <ys2833@columbia.edu>
Date:   Mon Oct 30 03:34:29 2017 -0400
entry point to gameobj main

Author: Steven <ys2833@columbia.edu>
Date:   Mon Oct 30 03:33:00 2017 -0400
generate gameobj code

Author: Steven <ys2833@columbia.edu>
Date:   Mon Oct 30 03:25:49 2017 -0400
```

```
semant for gameobjs

Author: Steven <ys2833@columbia.edu>
Date:   Mon Oct 30 01:33:42 2017 -0400
game_obj -> gameobj

Author: Steven <ys2833@columbia.edu>
Date:   Mon Oct 30 03:29:58 2017 -0400
float/global init changes to codegen

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 2 20:49:43 2017 -0400
c++11

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 2 12:49:27 2017 -0400
print float not a constant

Author: Steven <ys2833@columbia.edu>
Date:   Thu Nov 2 01:36:49 2017 -0400
tests work

Author: Steven <ys2833@columbia.edu>
Date:   Mon Oct 30 00:53:30 2017 -0400
add end_game

Author: Steven <ys2833@columbia.edu>
Date:   Sun Oct 29 03:52:29 2017 -0400
removed hard-coded graphics calls

Author: Steven <ys2833@columbia.edu>
Date:   Sun Oct 29 03:47:37 2017 -0400
external graphics fns

Author: Steven <ys2833@columbia.edu>
Date:   Sun Oct 29 03:40:11 2017 -0400
sprite, sound types

Author: Steven <ys2833@columbia.edu>
Date:   Sun Oct 29 03:31:56 2017 -0400
print float

Author: Steven <ys2833@columbia.edu>
Date:   Sun Oct 29 03:31:33 2017 -0400
float literal

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 04:26:42 2017 -0400
```

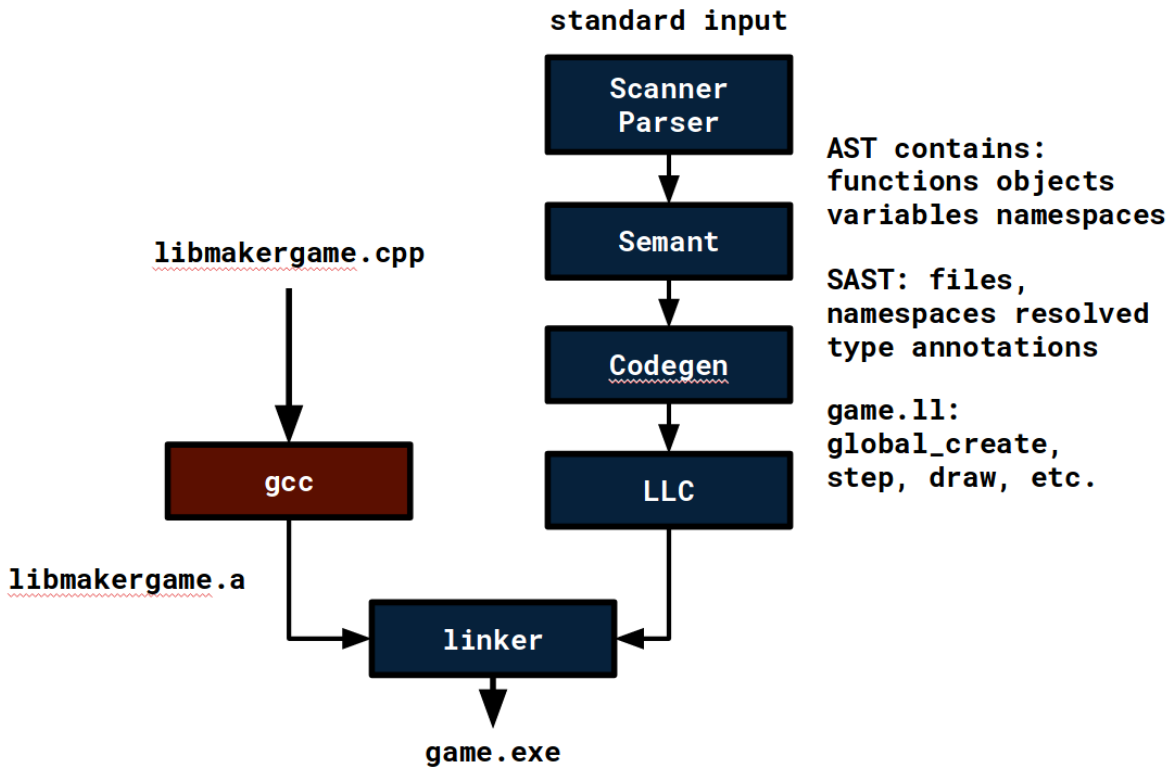
```
extern function keyword; duplicate function checks

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 03:56:03 2017 -0400
fixed rebase

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 00:28:20 2017 -0400
runtime functions

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 00:28:10 2017 -0400
gitignore

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 25 21:45:16 2017 -0400
fn_ prefix for functions, open window

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 03:08:30 2017 -0400
special character tests

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 02:51:15 2017 -0400
scanner fix

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 02:50:25 2017 -0400
tab

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 02:32:25 2017 -0400
tests

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 02:32:18 2017 -0400
strings

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 02:47:34 2017 -0400
raise failure -> failwith, \t to spaces

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 02:43:04 2017 -0400
better gitignore

Author: Steven <ys2833@columbia.edu>
Date:   Sat Oct 28 00:41:13 2017 -0400
```

```
.gitignore

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 25 03:03:06 2017 -0400
fn main entry point for now

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 25 02:59:45 2017 -0400
uncomment top level

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 25 02:59:36 2017 -0400
add 'not implemented' branches so functions compile

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 25 02:47:39 2017 -0400
gitignore

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 25 02:34:51 2017 -0400
formatting codegen

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 25 02:56:52 2017 -0400
semant handles fn blocks, game objects

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 25 02:54:19 2017 -0400
string type -> char type

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 25 00:55:46 2017 -0400
indenting and comments

Author: Steven <ys2833@columbia.edu>
Date:   Sun Oct 15 03:00:32 2017 -0400
directory structure, building, merlin

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 11 23:20:13 2017 -0400
prevent multiple event decls of same type

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 11 22:52:13 2017 -0400
game objects

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 11 20:51:35 2017 -0400
```

```
line comments

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 11 20:41:36 2017 -0400
only ast

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 11 20:40:52 2017 -0400
sprite and sound

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 11 20:04:02 2017 -0400
fewer errors in string/float

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 11 20:03:26 2017 -0400
operators

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 11 20:02:51 2017 -0400
ignore

Author: Steven <ys2833@columbia.edu>
Date:   Wed Oct 11 19:01:38 2017 -0400
floats and strings

Author: Steven <ys2833@columbia.edu>
Date:   Sun Oct 8 20:46:17 2017 -0400
gitignore

Author: Steven <ys2833@columbia.edu>
Date:   Sun Oct 8 20:30:51 2017 -0400
parsing arrays

Author: Steven <ys2833@columbia.edu>
Date:   Sun Oct 8 20:16:01 2017 -0400
MicroC starter
```

Architectural Design

The compiler architecture is largely the same as that of MicroC, with a few extra responsibilities for the semantic checker to handle namespace and file resolution and type annotation, instead of just verifying that its input is indeed a valid AST. Each member made contributions to each component in the compiler architecture.



The second addition is the runtime library, libmakergame, which gets linked with SFML and our generated code to produce the final executable.

The Runtime

From the LRM, the remaining features of the language per se that associate it with game programming are just the object definitions and the two opaque types whose names just happen to be associated with game resources. The compiler ultimately just generates a system that maintains and manipulates a collection of these eventful objects, and a few endpoints to execute steps in that system.

Externals & The Standard Library

One function of our runtime library, libmakergame, is to proivde implementations for common game operations that interface with libraries outside of MakerGame (e.g., SFML calls in functions compatible with external declarations in our generated code). Here is a sample listing of libmakergame's functions:

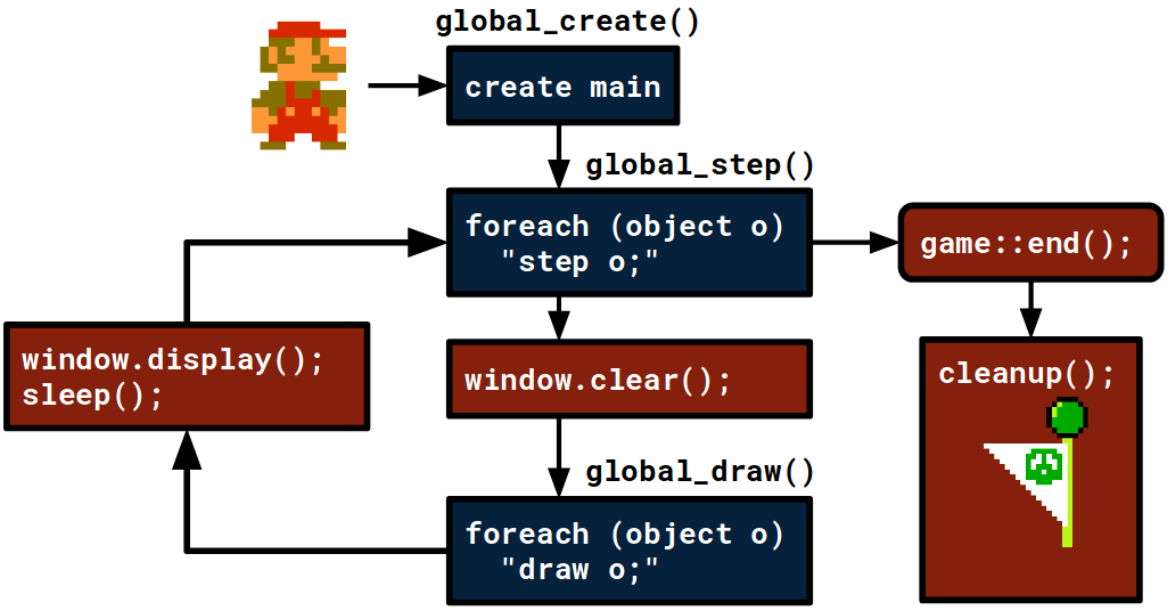
1. Printing
   ```
   void print(int x); void printb(bool b);
   void print_float(double x); void printstr(char *x);
   ```
2. Resource loading, inspection and utilization
   ```
   void *load_image(char *filename); void *load_sound(char *filename);
   void draw_sprite(void *spr, int x, int y); void play_sound(void *snd);
   int sprite_width(void *spr); int sprite_width(void *spr);
   void print_float(double x); void printstr(char *x);
   ```
3. Game settings and control
   ```
   void set_window_size(int w, int h);
   void set_window_clear(int r, int g, int b);
   void end_game();
   ```

Programmers can choose to refer to these functions directly with an `extern` declaration, but MakerGame's library system obviates this by declaring these external functions in a standard library. This standard library is defined in `std.mg` which, as the LRM states, is included in every namespace. This library contains several modules that access these functions (like `spr` or `window`) and wrap them in more palatable names like `std::window::clear(...)`, as well as other modules that define their own helper functions without relying on external function calls (e.g., specialized game object definitions and primitive hitbox calculations `std::game`, composite mathematical operations in `std::math`). The complete contents are available in the appendix.

So as to avoid naming conflicts between object methods, global functions, and external functions, we employ a name mangling scheme that tags a method's owner (or absence thereof), while preserving external function names.

Execution Flow

The second purpose of libmakergame is to provide an entry point and scaffolding for the subsequent game loop, represented by the diagram below:



The runtime envionment calls requires three external functions to be defined: global_create, global_step, and global_draw. These are provided by our generated code, and our compiler implements these functions with essentially the behaviour in the blue boxes. In between calls to our generated `global_*` functions, the runtime performs SFML library calls to handle rendering and frame limits. It is worth noting that during the execution of these `global_*` functions, other functions in libmakergame may be called (e.g., to get input).

The Object Collection

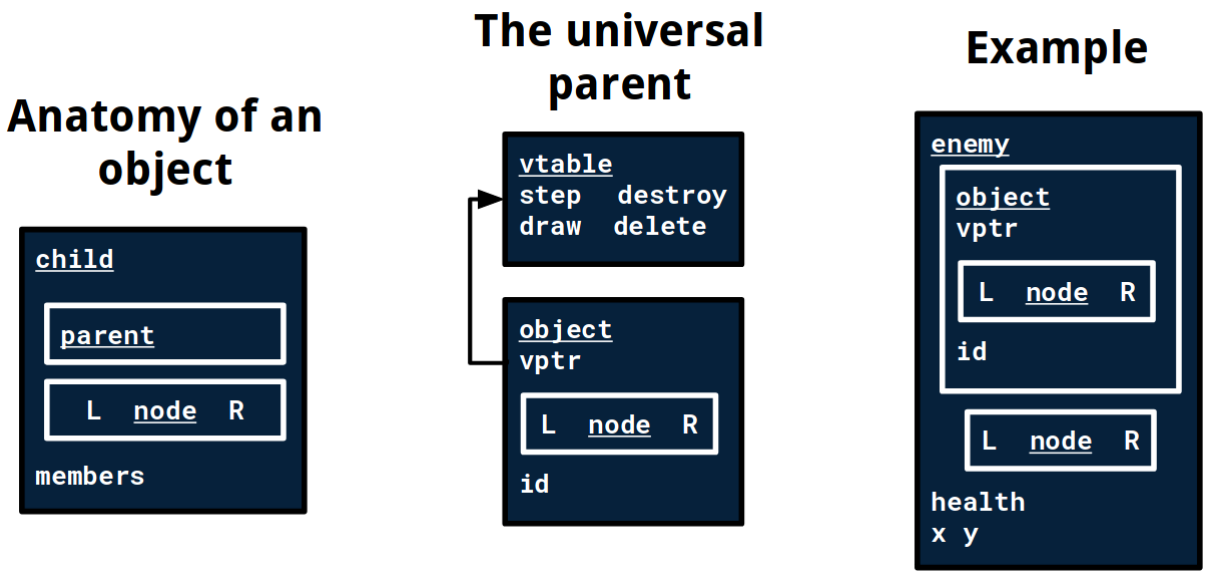
For each object type in MakerGame, a struct definition is generated that contains

1. A structure for this type's parent
2. A linked list node
3. The members of this object type

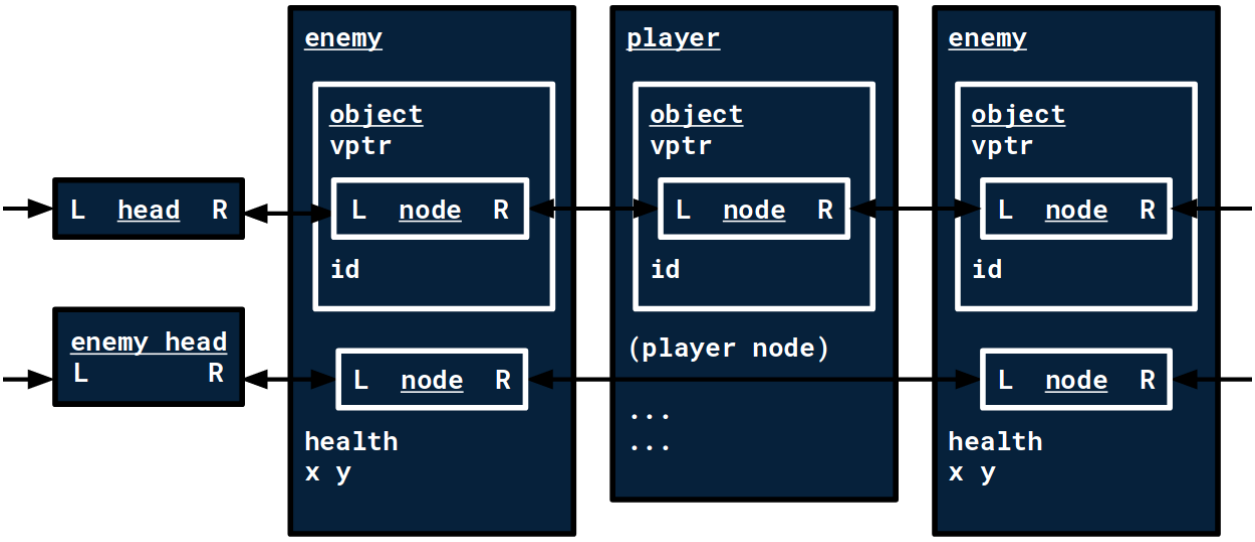
The generic object type has a slightly different specification:

1. A pointer to the virtual table (a list of four pointers, for each event)
2. A linked list node
3. An ID number

It's worth noting that each structure of a particular object type then participates in as many linked lists as the length of its ancestral inheritance chain, by design. The diagram below illustrates.



Completely independent from the external library, the primary data structure used to keep track of all the entities created and destroyed and enables the `foreach` loop is the object linked list hierarchy. This hierarchy contains a circular linked list for each object type. The diagram below demonstrates.



Iteration through a particular type of object then also includes descendants of those objects, and automatically skips non-descendants, with no additional runtime type information. Creation is then implemented as adding to the linked list, and destruction a lazy removal until the end of the top-level/global step iteration. (The structure actually has a few extra auxiliary nodes to ensure successful traversal even if, for example, the node currently being traversed is destroyed).

Test Plan

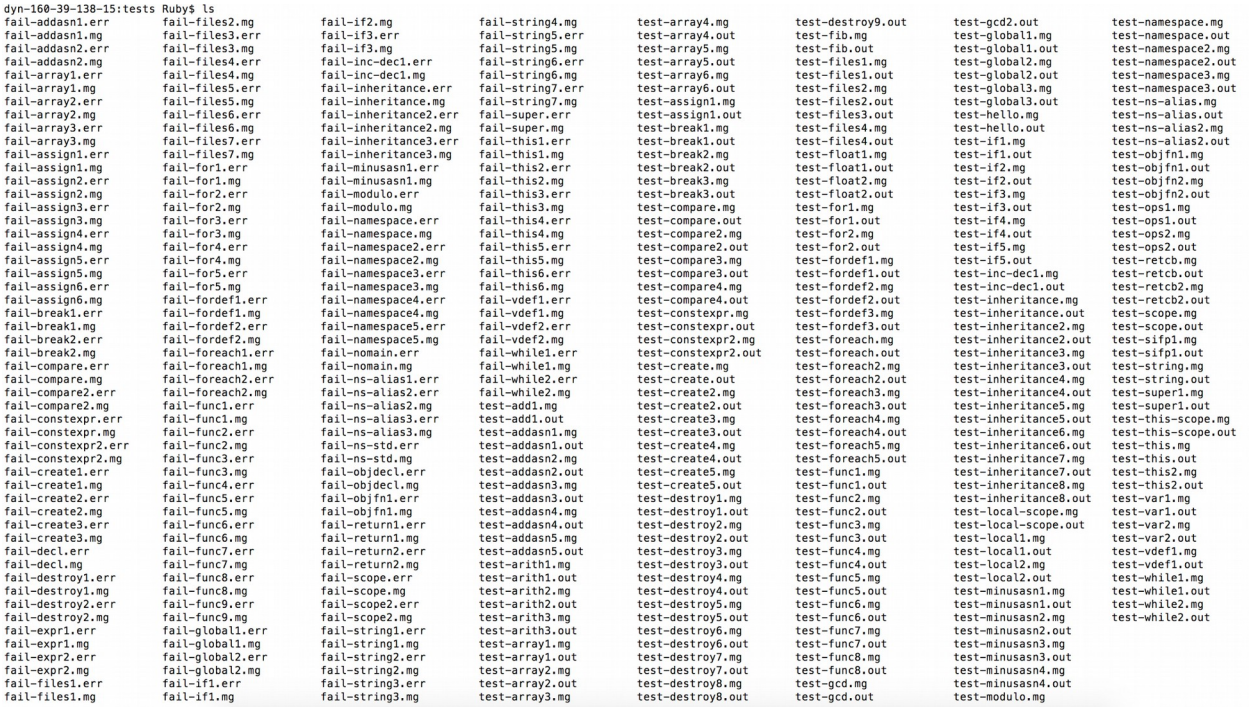
Testing Suite

For testing, we practiced test-driven development, so that we added tests every time we implement a new feature for our compiler.

Our test suite involves both success tests and fail tests. The success tests ensure that the correct programs run as we expect it to behave by comparing the actual print outputs to expected output. The fail tests makes sure that when incorrect program is written, the compiler can detect the error and throw appropriate error messages. We named our success tests using prefix “test”, and our fail tests using prefix “fail”.

We roughly chose these tests based on our expectation of how each feature should behave in regular cases and certain edge cases, and additionally added tests according to the different logical branches of code we add.

The following are the tests we wrote, and they are under tests/ directory.

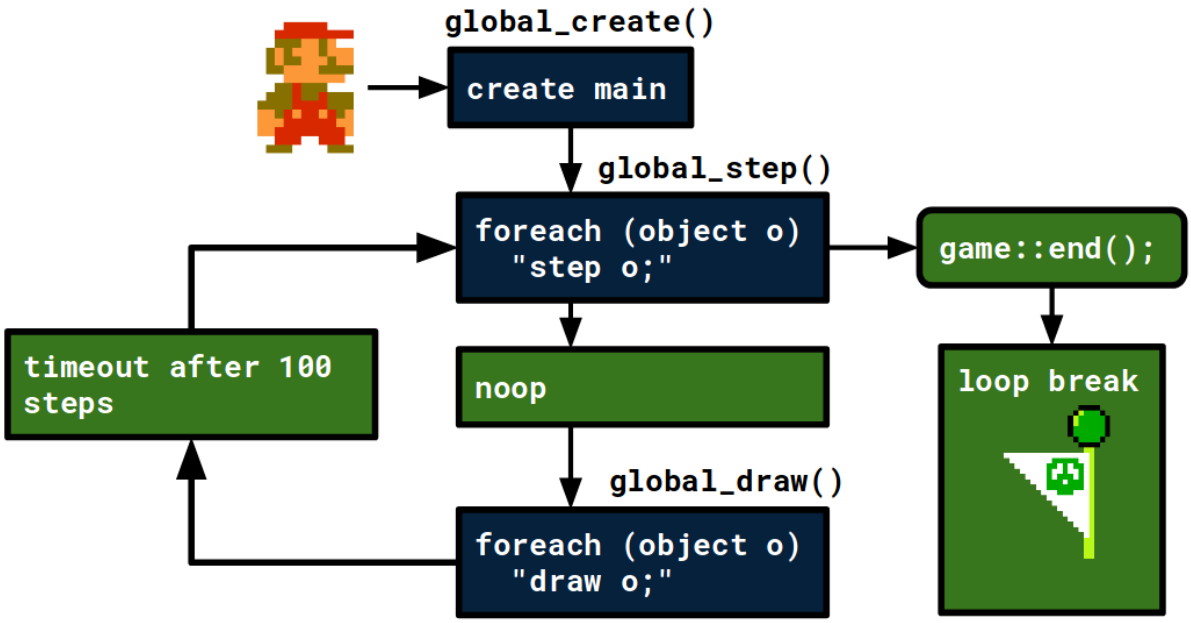


```
dyn-160-39-138-15:tests Ruby$ ls
fail-addasn1.err        fail-files2.mg          fail-if2.mg             fail-string4.mg         test-array4.mg          test-destroy9.out       test-gcd2.out           test-namespace.mg
fail-addasn1.mg         fail-files3.err         fail-if3.err            fail-string5.err        test-array4.out         test-fib.mg             test-global1.mg         test-namespace.out
fail-addasn2.err        fail-files3.mg          fail-if3.mg             fail-string5.mg         test-array5.mg          test-fib.out            test-global1.out        test-namespace2.mg
fail-addasn2.mg         fail-files4.err         fail-inc-dec1.err       fail-string6.err        test-array5.out         test-files1.mg          test-global2.mg         test-namespace2.out
fail-array1.err         fail-files4.mg          fail-inc-dec1.mg        fail-string6.mg         test-array6.mg          test-files1.out         test-global2.out        test-namespace3.mg
fail-array1.mg          fail-files5.err         fail-inheritance.err    fail-string7.err        test-array6.out         test-files2.mg          test-global3.mg         test-namespace3.out
fail-array2.err         fail-files5.mg          fail-inheritance.mg     fail-string7.mg         test-assign1.mg         test-files2.out         test-global3.out        test-ns-alias.mg
fail-array2.mg          fail-files6.err         fail-inheritance2.err   fail-super.err          test-assign1.out        test-files3.out         test-hello.mg           test-ns-alias.out
fail-array3.err         fail-files6.mg          fail-inheritance2.mg    fail-super.mg           test-break1.mg          test-files4.mg          test-hello.out          test-ns-alias2.mg
fail-array3.mg          fail-files7.err         fail-inheritance3.err   fail-this1.err          test-break1.out         test-files4.out         test-if1.mg             test-ns-alias2.out
fail-assign1.err        fail-files7.mg          fail-inheritance3.mg    fail-this1.mg           test-break2.mg          test-float1.mg          test-if1.out            test-objfn1.mg
fail-assign1.mg         fail-for1.err           fail-minusasn1.err      fail-this2.err          test-break2.out         test-float1.out         test-if2.mg             test-objfn1.out
fail-assign2.err        fail-for1.mg            fail-minusasn1.mg       fail-this2.mg           test-break3.mg          test-float2.mg          test-if2.out            test-objfn2.mg
fail-assign2.mg         fail-for2.err           fail-modulo.err         fail-this3.err          test-break3.out         test-float2.out         test-if3.mg             test-objfn2.out
fail-assign3.err        fail-for2.mg            fail-modulo.mg          fail-this3.mg           test-compare.mg         test-for1.mg            test-if3.out            test-ops1.mg
fail-assign3.mg         fail-for3.err           fail-namespace.err      fail-this4.err          test-compare.out        test-for1.out           test-if4.mg             test-ops1.out
fail-assign4.err        fail-for3.mg            fail-namespace.mg       fail-this4.mg           test-compare2.mg        test-for2.mg            test-if4.out            test-ops2.mg
fail-assign4.mg         fail-for4.err           fail-namespace2.err     fail-this5.err          test-compare2.out       test-for2.out           test-if5.mg             test-ops2.out
fail-assign5.err        fail-for4.mg            fail-namespace2.mg      fail-this5.mg           test-compare3.mg        test-fordef1.mg         test-if5.out            test-retcb.mg
fail-assign5.mg         fail-for5.err           fail-namespace3.err     fail-this6.err          test-compare3.out       test-fordef1.out        test-inc-dec1.mg        test-retcb.out
fail-assign6.err        fail-for5.mg            fail-namespace3.mg      fail-this6.mg           test-compare4.mg        test-fordef2.mg         test-inc-dec1.out       test-retcb2.mg
fail-assign6.mg         fail-fordef1.err        fail-namespace4.err     fail-vdef1.err          test-compare4.out       test-fordef2.out        test-inheritance.mg     test-retcb2.out
fail-break1.err         fail-fordef1.mg         fail-namespace4.mg      fail-vdef1.mg           test-constexpr.mg       test-fordef3.mg         test-inheritance.out    test-scope.mg
fail-break1.mg          fail-fordef2.err        fail-namespace5.err     fail-vdef2.err          test-constexpr.out      test-fordef3.out        test-inheritance2.mg    test-scope.out
fail-break2.err         fail-fordef2.mg         fail-namespace5.mg      fail-vdef2.mg           test-constexpr2.mg      test-foreach.mg         test-inheritance2.out   test-sifp1.mg
fail-break2.mg          fail-foreach1.err       fail-nomain.err         fail-while1.err         test-constexpr2.out     test-foreach.out        test-inheritance3.mg    test-sifp1.out
fail-compare.err        fail-foreach1.mg        fail-nomain.mg          fail-while1.mg          test-create.mg          test-foreach2.mg        test-inheritance3.out   test-string.mg
fail-compare.mg         fail-foreach2.err       fail-ns-alias1.err      fail-while2.err         test-create.out         test-foreach2.out       test-inheritance4.mg    test-string.out
fail-compare2.err       fail-foreach2.mg        fail-ns-alias2.err      fail-while2.mg          test-create2.mg         test-foreach3.mg        test-inheritance4.out   test-super1.mg
fail-compare2.mg        fail-func1.err          fail-ns-alias2.mg       test-add1.mg            test-create2.out        test-foreach3.out       test-inheritance5.mg    test-super1.out
fail-constexpr.err      fail-func1.mg           fail-ns-alias3.err      test-add1.out           test-create3.mg         test-foreach4.mg        test-inheritance5.out   test-this-scope.mg
fail-constexpr.mg       fail-func2.err          fail-ns-alias3.mg       test-addasn1.mg         test-create3.out        test-foreach4.out       test-inheritance6.mg    test-this-scope.out
fail-constexpr2.err     fail-func2.mg           fail-ns-std.err         test-addasn1.out        test-create4.mg         test-foreach5.mg        test-inheritance6.out   test-this.mg
fail-constexpr2.mg      fail-func3.err          fail-ns-std.mg          test-addasn2.mg         test-create4.out        test-foreach5.out       test-inheritance7.mg    test-this.out
fail-create1.err        fail-func3.mg           fail-objdecl.err        test-addasn2.out        test-create5.mg         test-func1.mg           test-inheritance7.out   test-this2.mg
fail-create1.mg         fail-func4.err          fail-objdecl.mg         test-addasn3.mg         test-create5.out        test-func1.out          test-inheritance8.mg    test-this2.out
fail-create2.err        fail-func5.err          fail-objfn1.err         test-addasn3.out        test-destroy1.mg        test-func2.mg           test-inheritance8.out   test-var1.mg
fail-create2.mg         fail-func5.mg           fail-objfn1.mg          test-addasn4.mg         test-destroy1.out       test-func2.out          test-local-scope.mg     test-var1.out
fail-create3.err        fail-func6.err          fail-return1.err        test-addasn4.out        test-destroy2.mg        test-func3.mg           test-local-scope.out    test-var2.mg
fail-create3.mg         fail-func6.mg           fail-return1.mg         test-addasn5.mg         test-destroy2.out       test-func3.out          test-local1.mg          test-var2.out
fail-decl.err           fail-func7.err          fail-return2.err        test-addasn5.out        test-destroy3.mg        test-func4.mg           test-local1.out         test-vdef1.mg
fail-decl.mg            fail-func7.mg           fail-return2.mg         test-arith1.mg          test-destroy3.out       test-func4.out          test-local2.mg          test-vdef1.out
fail-destroy1.err       fail-func8.err          fail-scope.err          test-arith1.out         test-destroy4.mg        test-func5.mg           test-local2.out         test-while1.mg
fail-destroy1.mg        fail-func8.mg           fail-scope.mg           test-arith2.mg          test-destroy4.out       test-func5.out          test-minusasn1.mg       test-while1.out
fail-destroy2.err       fail-func9.err          fail-scope2.err         test-arith2.out         test-destroy5.mg        test-func6.mg           test-minusasn1.out      test-while2.mg
fail-destroy2.mg        fail-func9.mg           fail-scope2.mg          test-arith3.mg          test-destroy5.out       test-func6.out          test-minusasn2.mg       test-while2.out
fail-expr1.err          fail-global1.err        fail-string1.err        test-arith3.out         test-destroy6.mg        test-func7.mg           test-minusasn2.out
fail-expr1.mg           fail-global1.mg         fail-string1.mg         test-array1.mg          test-destroy6.out       test-func7.out          test-minusasn3.mg
fail-expr2.err          fail-global2.err        fail-string2.err        test-array1.out         test-destroy7.mg        test-func8.mg           test-minusasn3.out
fail-expr2.mg           fail-global2.mg         fail-string2.mg         test-array2.mg          test-destroy7.out       test-func8.out          test-minusasn4.mg
fail-files1.err         fail-if1.err            fail-string3.err        test-array2.out         test-destroy8.mg        test-gcd.mg             test-minusasn4.out
fail-files1.mg          fail-if1.mg             fail-string3.mg         test-array3.mg          test-destroy8.out       test-gcd.out            test-modulo.mg
```

We used the MicroC test script provided to automate our tests.

libtestergame.cpp vs libmakergame.cpp

libmakergame.cpp is the runtime library used in the actual game source code (e.g. the tests under demo/). In unit testing however, it is annoying to have windows pop up for each test, and library functions such as “load_sound” and “draw_sprite” become unnecessary. Thus, we replaced the runtime in libmakergame.cpp with a stub runtime for unit testing that does not communicate with (or even link to) SFML in its game loop or its function implementations. For example, load_sound just returns a null pointer.



This way, we were able to more quickly and in an isolated manner test the internal runtime and execution of our generated code.

Notes

The source for our test external library is given in the appendix. Two demos with their corresponding LLVM IR output are also given.

Lessons Learned

Steven

I really enjoyed working on this project, having previously had a bit of OCaml, game development, software engineering experience. I was also very fortunate to have a lot of time this semester for each of my classes. Programming languages are one thing from the user perspective, but from the translator side, they're a whole different beast, and it was very exciting to get a glimpse of the underside and even start to empathize with compiler devs. Getting another chance to do something big in OCaml is always fun too, especially when the language is so suited to the domain. In building this C-GameMaker hybrid, I was humbled by the difficulties of integrating and tradeoffs between game and programming features. Understanding LLVM's bindings with OCaml and its IR was also tough, but nonetheless valuable in understanding an important layer in a compiler architecture.

I think we started sufficiently early, although very much at different paces, and if I could do the assignment again, I would hope to have the team a little more in sync or with better communication so each of us could have the confidence to contribute more equally.

Yuncheng

It was pretty interesting to design and implement a new language. We were trying to think from a game developer's perspective what he or she want when making a game. Combining the theoretical material we learned in class into a real programming language was exciting. I think it is important to keep track of the project and set the milestone. Since it was a semester-long project, having a clear plan and start early was really helpful. So we wrote a doc to track the progress of our project which became helpful through the semester. At the same time, keeping communicating with teammates about the implementation and design would be crucial to make sure everyone was on the same page.

Cindy

Do research and look at the projects from past semesters. It really helps you to find out what you are passionate to build and what available options are out there. Once you decide on what you are trying to build, be ambitious about the functionalities you'd like your language to have. As the semester goes on and you gain more background in class, you will figure out what features should not be included, and what additional features should be added. Then you could adjust your plan accordingly.

Appendix

Not every file heading is tagged with its contributors; in that case, at least one of its parent headings is.

Compiler code (Steven, Cindy, Yuncheng)

filename.c (for UNIX realpath) (from core_filename.ml in JS's Core)

```
#include <limits.h>
#include <caml/alloc.h>

CAMLprim value unix_realpath(value v_path) {
  char *path = String_val(v_path);
  char *res = realpath(path, NULL);
  value v_res;
  if (res == NULL) {
    v_res = caml_copy_string("");
  }
  else {
    v_res = caml_copy_string(res);
    free(res);
  }
  return v_res;
}
```

file.ml (for UNIX realpath)

```
include Filename
external unix_realpath : string -> string = "unix_realpath"
```

```
let realpath = function
  | "" -> raise Not_found
  | _ as p -> p

let resolve_file f curr_dir =
  try
    (* Attempt to resolve filenames in the following fashion:
       1) If the path is absolute, load that.
       2) If the path is relative, try:
          a) The current directory of the file loading you.
          c) MAKERGAME_PATH
    *)
    if not (Filename.is_relative f) then realpath f
    else
      let cdir_f = Filename.concat curr_dir f in
      let sdir_f = Filename.concat (Sys.getenv "MAKERGAME_PATH") f in
      if Sys.file_exists cdir_f then realpath cdir_f
      else if Sys.file_exists sdir_f then realpath sdir_f
      else raise Not_found
  with
  | Unix.Unix_error (e, _, _) ->
    failwith ("unable to open file \'" ^ f ^ "\': \'" ^ Unix.error_message e
^ "\'")
  | Not_found | Sys_error _ ->
    failwith ("unable to open file \'" ^ f ^ "\'")

let dirname = Filename.dirname;;
```

ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)

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

type idop = Inc | Dec

type asnop = Addasn | Subasn | Multasn | Divasn

type uop = Neg | Not

(* Chain of namespaces, then ID *)
type id_chain = string list * string

type typ =
  | Void
  | Int
```

```
  | Bool
  | Float
  | String
  | Sprite
  | Sound
  | Object of id_chain          (* object type, in relative position of
namespaces *)
  | Arr of typ * int

module Var = struct
  type t = typ
  type decl = string * t
end

type expr =
  | Literal of int
  | BoolLit of bool
  | FloatLit of float
  | StringLit of string
  | NoneObject
  | ArrayLit of expr list
  | Id of id_chain
  | Conv of typ * expr * typ
  | Binop of expr * op * typ * expr
  | Asnop of expr * asnop * typ * expr
  | Unop of uop * typ * expr
  | Idop of idop * typ * expr
  | Assign of expr * expr
  | Subscript of expr * expr
  (* (LHS before period, name of LHS object type, RHS) *)
  | Member of expr * id_chain * string
  (* (LHS before period, name of LHS object type, RHS function name, actuals)
*)
  | MemberCall of expr * id_chain * string * expr list
  | Call of id_chain * expr list
  | Create of id_chain * expr list
  | Destroy of expr
  | Delete of expr              (* Remove the object without calling the
destroy event. *)
  | Noexpr

type stmt =
  | Decl of Var.decl
  | Vdef of Var.decl * expr
  | Block of block
  | Expr of expr
  | Break
  | Return of expr
```

```
  | If of expr * stmt * stmt
  | For of stmt * expr * expr * stmt
  | Foreach of Var.decl * stmt
  | While of expr * stmt
and block = stmt list

module Func = struct
  type t = {
    typ : typ;
    formals : Var.decl list;
    gameobj : string option;
    block : block option;
  }
  type decl = string * t

  let make typ formals gameobj block = { typ; formals; gameobj; block = (Some
block) }
end

module Gameobj = struct
  type t = {
    members : Var.decl list;
    methods : Func.decl list;
    events : Func.decl list;
    parent : id_chain option;
  }
  type decl = string * t

  let make name (members, methods, events) parent =
    (* Tag each method with this object name *)
    let add_gameobj (n, x) = n, { x with Func.gameobj = Some name } in
    let methods = List.map add_gameobj methods in
    let events  = List.map add_gameobj events in
    name, { members; methods; events; parent }

  let add_vdecl (vdecls, fdecls, edecls) vdecl =
    (vdecl :: vdecls, fdecls, edecls)

  let add_fdecl (vdecls, fdecls, edecls) fdecl =
    (vdecls, fdecl :: fdecls, edecls)

  let add_edecl (vdecls, fdecls, edecls) edecl =
    (vdecls, fdecls, edecl :: edecls)

  let generic =
    let _, obj = make "object" ([], [], []) None in obj
end
```

```
module Namespace = struct
  type concrete = {
    namespaces : decl list;
    usings     : (bool * string list) list; (* Private or not, namespace name
*)
    variables  : (Var.decl * expr) list;
    functions  : Func.decl list;
    gameobjs   : Gameobj.decl list;
  }
  (* a namespace could be an alias for another in the tree, with the same
values *)
  and t = Concrete of concrete | Alias of string list | File of string
  and decl = string * (bool * t)

  let make (variables, functions, gameobjs, namespaces, usings) =
    { variables; functions; gameobjs; namespaces; usings }

  let add_vdecl (vdecls, fdecls, odecls, ndecls, udecls) vdecl =
    (vdecl :: vdecls, fdecls, odecls, ndecls, udecls)

  let add_fdecl (vdecls, fdecls, odecls, ndecls, udecls) fdecl =
    (vdecls, fdecl :: fdecls, odecls, ndecls, udecls)

  let add_odecl (vdecls, fdecls, odecls, ndecls, udecls) odecl =
    (vdecls, fdecls, odecl :: odecls, ndecls, udecls)

  let add_ndecl (vdecls, fdecls, odecls, ndecls, udecls) ndecl =
    (vdecls, fdecls, odecls, ndecl :: ndecls, udecls)

  let add_udecl (vdecls, fdecls, odecls, ndecls, udecls) udecl =
    (vdecls, fdecls, odecls, ndecls, udecl :: udecls)

end

type program = {
  main : Namespace.concrete;
  (* TODO: handle matching files in different directories *)
  files : (string * Namespace.concrete) list
}



(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
```

```
  | Div -> "/"
  | Modulo -> "%"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"

let string_of_idop = function
    Inc -> "++"
  | Dec -> "--"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"

let string_of_chain (c, e) = (List.fold_left (fun s x -> s ^ x ^ "::") "" c)
^ e

let string_of_asnop = function
    Addasn -> "+="
  | Subasn -> "-="
  | Multasn -> "*="
  | Divasn -> "/="

let rec string_of_typ_ps = function
    Int -> "int", ""
  | Bool -> "bool", ""
  | Void -> "void", ""
  | Float -> "float", ""
  | Sprite -> "sprite", ""
  | Sound -> "sound", ""
  | Object (_, "object") -> "object", ""
  | Object obj_t -> string_of_chain obj_t, ""
  | Arr(typ, len) ->
    let pref, suf = string_of_typ_ps typ in
    pref, "[" ^ (string_of_int len) ^ "]" ^ suf
  | String -> "string", ""

let string_of_typ t =
  (* Split by prefix and suffix so array dims are in the right order *)
  let p, s = string_of_typ_ps t in
  p ^ s

let rec string_of_expr = function
```

```
    Literal(l) -> string_of_int l
  | StringLit(s) -> "\"" ^ s ^ "\""
  | FloatLit(f) -> string_of_float f
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | NoneObject -> "none"
  | ArrayLit l -> "[" ^ String.concat ", " (List.map string_of_expr l) ^ "]"
  | Id c -> string_of_chain c
  (* Conv (t, e, _) expresses a conversion of e to type t. _ tags e's
original type. *)
  | Conv (t, e, _) -> "(" ^ string_of_typ t ^ ")" ^ string_of_expr e
  | Asnop(l, o, _, r) ->
    string_of_expr l ^ " " ^ string_of_asnop o ^ " " ^ string_of_expr r
  | Binop(e1, o, _, e2) ->
    string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr e2
  | Idop(o, _, e) -> string_of_idop o ^ string_of_expr e
  | Unop(o, _, e) -> string_of_uop o ^ string_of_expr e
  | Assign(l, r) -> string_of_expr l ^ " = " ^ string_of_expr r
  | Call(f, el) ->
    (string_of_chain f) ^ "(" ^ String.concat ", " (List.map string_of_expr
el) ^ ")"
  | Subscript(e, ind) -> string_of_expr e ^ "[" ^ string_of_expr ind ^ "]"
  | Member(e, _, s) -> "(" ^ (string_of_expr e) ^ ")." ^ s
  | MemberCall(e, _, f, el) -> "(" ^ (string_of_expr e) ^ ")." ^
string_of_expr (Call(([], f), el))
  | Create (c, args) ->
    (match args with
     | [] -> "create " ^ string_of_chain c
     | _ -> "create " ^ string_of_expr(Call(c, args)))
  | Destroy o -> "destroy " ^ (string_of_expr o)
  | Delete o -> "delete " ^ (string_of_expr o)
  | Noexpr -> ""

let string_of_vdecl (id, t) = let p, s = string_of_typ_ps t in p ^ " " ^ id ^
s
let string_of_global_vdecl (d, e) =
  match e with
  | Noexpr -> string_of_vdecl d
  | _ -> string_of_vdecl d ^ " = " ^ string_of_expr e

let rec string_of_stmt = function
  | Decl d -> string_of_vdecl d ^ ";"
  | Vdef(d, e) -> string_of_vdecl d ^ " = " ^ string_of_expr e ^ ";"
  | Block(blk) -> string_of_block blk
  | Expr(expr) -> string_of_expr expr ^ ";"
  | Break -> "break;"
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";"
  | 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(s1, e2, e3, s2) ->
    "for (" ^ string_of_stmt s1 ^ " " ^ string_of_expr e2 ^ "; " ^
    string_of_expr e3  ^ ") " ^ string_of_stmt s2
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
  | Foreach(d, s) ->
    "foreach (" ^ string_of_vdecl d ^ ") " ^ string_of_stmt s
and string_of_block block =
  "{\n" ^ String.concat "\n" (List.map string_of_stmt block) ^ "\n}\n"

let string_of_edecl (name, func) =
  let open Func in
  let block_str =
    match func.block with
    | None -> assert false
    | Some block -> string_of_block block
  in
  let arg_str = match func.formals with
    | [] -> ""
    | _ as args -> "(" ^ String.concat ", " (List.map string_of_vdecl args) ^
")"
  in
  "event " ^ name ^ arg_str ^ "\n" ^ block_str

let string_of_fdecl (name, func) =
  let open Func in
  let prefix, suffix =
    match func.block with
    | None -> "extern ", ""
    | Some block -> "", string_of_block block
  in
  prefix ^ string_of_vdecl (name, func.typ) ^ "(" ^
  String.concat ", " (List.map string_of_vdecl func.formals) ^ ")\n" ^ suffix

let with_semi string_of x = string_of x ^ ";";;

let string_of_gameobj (name, obj) =
  let open Gameobj in
  let par_str = match obj.parent with
    | None | Some (_, "object")-> ""
    | Some p -> " : " ^ string_of_chain p
  in
  "object " ^ name ^ par_str ^ " {\n" ^
  String.concat "\n" (List.map (with_semi string_of_vdecl) obj.members) ^
"\n" ^
  String.concat "" (List.map string_of_fdecl obj.methods) ^ "\n" ^
```

```
  String.concat "" (List.map string_of_edecl obj.events) ^ "\n" ^
  "}\n"

let string_of_udecl (priv, ns) =
  "using " ^ (if priv then "private " else "") ^ (String.concat "::" ns) ^ ";"
;;

let rec string_of_concrete_ns
    { Namespace.variables; functions; gameobjs; namespaces; usings } =
  String.concat "\n" (List.map string_of_udecl usings) ^ "\n" ^
  String.concat "\n" (List.map (with_semi string_of_global_vdecl) variables)
^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl functions) ^
  String.concat "\n" (List.map string_of_gameobj gameobjs) ^
  String.concat "\n" (List.map string_of_ns_decl namespaces)
and string_of_ns_decl (name, (is_private, ns)) =
  let open Namespace in
  let pref = if is_private then "private namespace " else "namespace " in
  match ns with
  | Alias chain -> pref ^ name ^ " = " ^ String.concat "::" chain ^";\n"
  | File f      -> pref ^ name ^ " = open \"" ^ f ^"\";\n"
  | Concrete n  -> pref ^ name ^ " {\n" ^ string_of_concrete_ns n ^ "\n}\n"

let string_of_program { main; files = _ } = string_of_concrete_ns main
```

llast.ml

```
(* Tree to keep track of llvalues for various symbols in a program *)
module L = Llvm
module StringMap = Map.Make(String)

type func = {
  value: L.llvalue;
  typ: L.lltype;
  return: Ast.typ;
  gameobj: string option;
}

type gameobj = {
  gtyp: L.lltype;
  head: L.llvalue;
  methods: func StringMap.t;
  events: func StringMap.t;
  vtable: L.llvalue;            (* TODO: events are replaced even if they're
not defined in child class. *)
  semant: Ast.Gameobj.t;
  (* TODO: here the ast obj is stored with the llobj. possibly in another
place
```

```
     we do two separate lookups. unify *)
}

type concrete = {
  variables: (L.llvalue * Ast.typ) StringMap.t;
  functions: func StringMap.t;
  gameobjs: gameobj StringMap.t;
  namespaces: namespace StringMap.t;
}
and namespace = Concrete of concrete | Alias of string list | File of string
```

scanner.mll

```
{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
  | "/*"     { block_comment lexbuf }           (* Comments *)
  | "//"     { line_comment lexbuf }
  (* grouping *)
  | '['      { LBRACK }
  | ']'      { RBRACK }
  | '('      { LPAREN }
  | ')'      { RPAREN }
  | '{'      { LCURLY }
  | '}'      { RCURLY }
  (* separators *)
  | ';'      { SEMI }
  | ','      { COMMA }
  | '.'      { PERIOD }
  | "::"     { DBCOLON }
  | ":"      { COLON }
  (* arithmetic and logical *)
  | '+'      { PLUS }
  | '-'      { MINUS }
  | '*'      { TIMES }
  | '/'      { DIVIDE }
  | '%'      { MODULO }
  | "++"     { INCREMENT }
  | "--"     { DECREMENT }
  | '='      { ASSIGN }
  | "+="     { ADDASN }
  | "-="     { MINUSASN }
  | "*="     { TIMEASN }
  | "/="     { DIVASN }
  | "=="     { EQ }
  | "!="     { NEQ }
  | '<'      { LT }
```

```
  | "<="     { LEQ }
  | ">"      { GT }
  | ">="     { GEQ }
  | "&&"     { AND }
  | "||"     { OR }
  | "!"      { NOT }
  (* control flow *)
  | "if"     { IF }
  | "else"   { ELSE }
  | "for"    { FOR }
  | "while"  { WHILE }
  | "foreach" { FOREACH }
  | "break"  { BREAK }
  | "return" { RETURN }
  (* datatypes *)
  | "int"    { INT }
  | "bool"   { BOOL }
  | "string" { STRING }
  | "float"  { FLOAT }
  | "void"   { VOID }
  | "sprite" { SPRITE }
  | "sound"  { SOUND }
  (* game keywords *)
  | "object" { OBJECT }
  | "none"   { NONE }
  | "event"  { EVENT }
  | "create" { CREATE }
  | "delete" { DELETE }
  | "destroy" { DESTROY }
  | "draw"   { DRAW }
  | "step"   { STEP }
  (* misc keywords *)
  | "namespace" { NAMESPACE }
  | "private" { PRIVATE }
  | "public" { PUBLIC }
  | "extern" { EXTERN }
  | "open"   { OPEN }
  | "using"  { USING }
  (* literals *)
  | "true"   { TRUE }
  | "false"  { FALSE }
  | ['0'-'9']+ as lxm { LITERAL(int_of_string lxm) }
  | ['0'-'9']+'.'['0'-'9']* as lxm { FLOATLIT(float_of_string lxm) }
  | '.'['0'-'9']+ as lxm { FLOATLIT(float_of_string lxm) }
  (* | ['0'-'9']+'e'['+''-']['0'-'9']+ as lxm { LITERAL(float_of_string
lxm) } *)
  (* | ['0'-'9']+'.'['0'-'9']*('e'['+''-']['0'-'9']+)? as lxm
{ LITERAL(float_of_string lxm) } *)
```

```
  (* | '.'['0'-'9']+('e'['+''-']['0'-'9']+)? as lxm { LITERAL(float_of_string
lxm) } *)
  | ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
  | '"' { string_literal "" lexbuf }
  | eof { EOF }
  | _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and line_comment = parse
    '\n' { token lexbuf }
  | _    { line_comment lexbuf }

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

and string_literal accum = parse
  | '\\' { escaped_string_literal accum lexbuf }
  | '"' { STRLIT (Scanf.unescaped accum) }
  | [^'\"' '\\'] as c { string_literal (accum ^ (Char.escaped c)) lexbuf }

and escaped_string_literal accum = parse
    (* only n, \, quote escape characters for now *)
  | ['\\' '\"' 'n'] as c { string_literal (accum ^ "\\" ^ (String.make 1 c))
lexbuf }
  | _ as c { failwith ("unsupported escape character \\" ^ Char.escaped c) }
```

parser.mly

```
%{
open Ast
%}

%token SEMI LPAREN RPAREN LBRACK RBRACK LCURLY RCURLY COMMA PERIOD COLON
DBCOLON
%token PLUS MINUS TIMES DIVIDE MODULO ASSIGN NOT
%token INCREMENT DECREMENT
%token ADDASN MINUSASN TIMEASN DIVASN
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR
%token BREAK RETURN IF ELSE FOR WHILE FOREACH
%token INT BOOL FLOAT STRING SPRITE SOUND VOID
%token OBJECT NONE EVENT CREATE DESTROY DRAW STEP DELETE
%token PRIVATE PUBLIC NAMESPACE USING EXTERN OPEN
%token <int> LITERAL
%token <string> ID
%token <string> STRLIT
%token <float> FLOATLIT
%token EOF
```

```
%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN ADDASN MINUSASN TIMEASN DIVASN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%right CREATE DESTROY DELETE
%left PLUS MINUS
%left TIMES DIVIDE MODULO
%right NOT NEG
%left INCREMENT DECREMENT
%left PERIOD LBRACK RBRACK
%left DBCOLON

%start program
%type <Ast.program> program

%%

program: decls EOF { { main = Namespace.make $1; files = [] } }

ndecl:
 | PUBLIC ndecl_base  { let n, ns = $2 in n, (false, ns) }
 | ndecl_base         { let n, ns = $1 in n, (false, ns) }
 | PRIVATE ndecl_base { let n, ns = $2 in n, (true, ns) }

ndecl_base:
 | NAMESPACE ID LCURLY decls RCURLY { $2, Namespace.Concrete (Namespace.make
$4) }
 | NAMESPACE ID ASSIGN id_chain SEMI { $2, Namespace.Alias (fst $4 @ [snd
$4]) }
 | NAMESPACE ID ASSIGN OPEN STRLIT SEMI { $2, Namespace.File $5 }

udecl:
 | USING id_chain SEMI         { (false, fst $2 @ [snd $2]) }
 /* | USING PRIVATE id_chain SEMI { (true, fst $3 @ [snd $3]) } */ /* no
private using for now */

decls:
   /* nothing */ { [],[],[],[],[] }
 | global_vdecl decls { Namespace.add_vdecl $2 $1 }
 | udecl decls { Namespace.add_udecl $2 $1 }
 | fdecl decls { Namespace.add_fdecl $2 $1 }
 | odecl decls { Namespace.add_odecl $2 $1 }
 | ndecl decls { Namespace.add_ndecl $2 $1 }

global_vdecl:
```

```
 | global_bind SEMI { $1, Noexpr }
 | global_bind ASSIGN expr SEMI { $1, $3 }

code_block:
   LCURLY stmt_list RCURLY { $2 }

array_list:
 /* nothing */ { [] }
 | LBRACK LITERAL RBRACK array_list { $2 :: $4 }

/* arrays dimensions are done RtL so we fold right*/
bind:
 | global_bind { $1 }
 | OBJECT ID array_list
   { List.fold_right (fun l (name, typ) -> (name, Arr(typ, l))) $3 ($2,
Object([], "object")) }

global_bind:
 | global_typ ID array_list
   { List.fold_right (fun l (name, typ) -> (name, Arr(typ, l))) $3 ($2, $1) }


fdecl:
 | EXTERN bind LPAREN formals_opt RPAREN SEMI
     { let name, typ = $2 in name, { Func.typ; formals = $4; gameobj = None;
block = None } }
 | bind LPAREN formals_opt RPAREN code_block
     { let name, typ = $1 in name, Func.make typ $3 None $5 }

event:
 | EVENT CREATE LPAREN formals_opt RPAREN code_block
   { "create", Func.make Void $4 None $6 }
 | EVENT CREATE  code_block { "create",  Func.make Void [] None $3 }
 | EVENT STEP    code_block { "step",    Func.make Void [] None $3 }
 | EVENT DESTROY code_block { "destroy", Func.make Void [] None $3 }
 | EVENT DRAW    code_block { "draw",    Func.make Void [] None $3 }

odecl:
 | OBJECT ID COLON id_chain LCURLY odecl_body RCURLY
     { Gameobj.make $2 $6 (Some $4) }
 | OBJECT ID LCURLY odecl_body RCURLY
     { Gameobj.make $2 $4 (Some ([], "object")) }

odecl_body:
    /* nothing */ { [], [], [] }
  | odecl_body vdecl { Gameobj.add_vdecl $1 $2 }
  | odecl_body fdecl { Gameobj.add_fdecl $1 $2 }
  | odecl_body event { Gameobj.add_edecl $1 $2 }
```

```
formals_opt:
    /* nothing */ { [] }
  | formal_list   { $1 }

formal_list:
    bind                   { [$1] }
  | bind COMMA formal_list { $1 :: $3 }

global_typ: /* excludes OBJECT, for shift/reduce conflict purposes */
    INT { Int }
  | BOOL { Bool }
  | SPRITE { Sprite }
  | SOUND { Sound }
  | VOID { Void }
  | FLOAT { Float }
  | STRING { String }
  | id_chain { Object($1) }

vdecl:
  | bind SEMI { $1 }

vdef:
  | bind ASSIGN expr SEMI { Vdef($1, $3) }

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

stmt:
    expr SEMI { Expr $1 }
  | vdecl { Decl $1 }
  | vdef { $1 }
  | RETURN expr_opt SEMI { Return $2 }
  | BREAK SEMI { Break }
  | code_block { Block($1) }
  | 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 (Expr($3), $5, $7, $9) }
  | FOR LPAREN vdef expr SEMI expr_opt RPAREN stmt
     { For($3, $4, $6, $8) }
  | WHILE LPAREN expr RPAREN stmt { While($3, $5) }
  | FOREACH LPAREN bind RPAREN stmt { Foreach($3, $5) }

expr_opt:
    /* nothing */ { Noexpr }
```

```
  | expr          { $1 }

expr:
  | LITERAL           { Literal($1) }
  | STRLIT            { StringLit($1) }
  | FLOATLIT          { FloatLit($1) }
  | TRUE              { BoolLit(true) }
  | FALSE             { BoolLit(false) }
  | NONE              { NoneObject }
  | id_chain          { Id($1) }
  | expr PERIOD ID    { Member($1, ([], ""), $3) }
  | expr PLUS   expr  { Binop($1, Add, Void,  $3) }
  | expr MINUS  expr  { Binop($1, Sub, Void,  $3) }
  | expr TIMES  expr  { Binop($1, Mult, Void, $3) }
  | expr DIVIDE expr  { Binop($1, Div, Void,  $3) }
  | expr MODULO expr  { Binop($1, Modulo, Void,  $3) }
  | expr EQ     expr  { Binop($1, Equal, Void,   $3) }
  | INCREMENT   expr  { Idop(Inc, Void, $2) }
  | DECREMENT   expr  { Idop(Dec, Void, $2)}
  | expr ADDASN expr  { Asnop($1, Addasn, Void, $3) }
  | expr MINUSASN expr { Asnop($1, Subasn, Void, $3) }
  | expr TIMEASN expr  { Asnop($1, Multasn, Void, $3) }
  | expr DIVASN expr  { Asnop($1, Divasn, Void, $3) }
  | expr NEQ    expr  { Binop($1, Neq, Void,  $3) }
  | expr LT     expr  { Binop($1, Less, Void, $3) }
  | expr LEQ    expr  { Binop($1, Leq, Void,  $3) }
  | expr GT     expr  { Binop($1, Greater, Void, $3) }
  | expr GEQ    expr  { Binop($1, Geq, Void,  $3) }
  | expr AND    expr  { Binop($1, And, Void,  $3) }
  | expr OR     expr  { Binop($1, Or, Void,   $3) }
  | MINUS expr %prec NEG { Unop(Neg, Void,   $2) }
  | NOT expr          { Unop(Not, Void, $2) }
  | expr ASSIGN expr  { Assign($1, $3) }
  | expr LBRACK expr RBRACK { Subscript($1, $3) }
  | LBRACK expr_list RBRACK { ArrayLit $2 }
  | id_chain LPAREN actuals_opt RPAREN { Call($1, $3) }
  | expr PERIOD ID LPAREN actuals_opt RPAREN { MemberCall($1, ([], ""), $3,
$5) }
  | CREATE id_chain                           { Create($2, []) }
  | CREATE id_chain LPAREN actuals_opt RPAREN { Create($2, $4) }
  | DESTROY expr      { Destroy($2) }
  | DELETE expr       { Delete($2) }
  | LPAREN expr RPAREN { $2 }

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

```
id_chain:
  | ID { [], $1 }
  | ID DBCOLON id_chain { let c, e = $3 in $1 :: c, e }

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

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

semant.ml

```
open Ast

module StringMap = Map.Make(String)

(* Raise an exception if the given list has a duplicate *)
let report_duplicate exceptf list =
  let rec helper = function
    | n1 :: n2 :: _ when n1 = n2 -> failwith (exceptf n1)
    | _ :: t -> helper t
    | [] -> ()
  in helper (List.sort compare list)

(* Raise an exception if a given binding is to a void type *)
let check_not_void exceptf = function
  | (n, Void) -> failwith (exceptf n)
  | _ -> ()

(* Add to the scope. But only if we're not using the identifier 'this' *)
(* TODO: mention where 'this'/super can be used. anywhere but assignment,
declaration as a regular obj identifier *)
let add_to_scope ?loc m (n, t) =
  let failstr s = ("cannot shadow or overwrite identifier '" ^ s ^ "'") in
  match loc, n with
  | None,   "super" -> failwith (failstr "super")
  | None,   "this"  -> failwith (failstr "this")
  | Some l, "super" -> failwith ((failstr "super") ^ " in " ^ l)
  | Some l, "this"  -> failwith ((failstr "this") ^ " in " ^ l)
  | _ -> StringMap.add n t m

(* Raise an exception if the two types are unequal. *)
let check_assign_strict lvaluet rvalue rvaluet err =
  if lvaluet = rvaluet then (lvaluet, rvalue)
  else failwith err
```

```
(* Build an AST given a resolved filename. For use in namespaces for file-
loading. *)
let ast_of_file f short_f =
  let channel =
    try open_in f
    with
    | Unix.Unix_error (e, _, _) ->
      failwith ("unable to open file \'" ^ f ^ "\' for namespace: \'" ^
Unix.error_message e ^ "\'")
    | Not_found | Sys_error _ ->
      failwith ("unable to open file \'" ^ f ^ "\' for namespace.")
  in
  try Parser.program Scanner.token (Lexing.from_channel channel)
  with Parsing.Parse_error -> failwith ("failed to parse file " ^ short_f)
;;

(* Access a namespace inside another namespace (top) through a chain. Prevent
   access to private namespaces.

   files should contain every file that could possibly be accessed in the
   chain. prev accumulates previous namespace accesses to detect loops. cname
   is a name for the chain in error messages. *)
let rec namespace_of_chain top prev files can_private chain cname =
  match chain with
  | [] -> top
  | hd :: tl ->
    (* Check for permissions and get the next namespace *)
    let inner_ns =
      let is_private, ns =
        try List.assoc hd top.Namespace.namespaces
        with Not_found ->
          failwith ("unrecognized namespace " ^ hd ^
                    " encountered when resolving " ^ cname)
      in
      if is_private && (not can_private)
      then failwith ("attempted access to private namespace " ^
                     hd ^ " in resolving " ^ cname)
      else ns
    in
    (* Check for loops using name reference. Update prev *)
    let prev' =
      if List.mem (top, chain) prev
      then failwith ("namespace " ^ cname ^ " never resolves")
      else (top, chain) :: prev
    in
    (* Recurse down depending on nature of next namespace *)
    let top', chain', private' =
      match inner_ns with
```

```
      | Namespace.Concrete c -> c, tl, false
      | Namespace.Alias a -> top, (a @ tl), true
      | Namespace.File f -> List.assoc f files, tl, false
    in
    namespace_of_chain top' prev' files private' chain' cname
;;

let namespace_scope namespace_of_chain nname =
  let rec helper chain can_private =
    let nname = List.fold_left (fun acc x -> acc ^ "::" ^ x) nname chain in
    let open Namespace in
    let ns = namespace_of_chain chain in
    let add_ns_scope (v, f) (priv, inner_chain) =
      match can_private, priv with
      (* TODO might not work with the way codegen works *)
      (* Don't include private usings *)
      (* | false, true -> v, f *)
      | _ ->
        let vs, fs = helper (chain @ inner_chain) false in
        StringMap.fold StringMap.add vs v, StringMap.fold StringMap.add fs f
    in
    let vscope, fscope =
      List.fold_left add_ns_scope (StringMap.empty, StringMap.empty) ns.usings
    in
    let global_binds = List.map fst ns.variables in
    let functions = ns.functions in
    List.fold_left (add_to_scope ~loc:(nname ^ " globals")) vscope
global_binds,
    List.fold_left (add_to_scope ~loc:(nname ^ " fn decls")) fscope functions
  in
  helper [] true
;;

(* Semantic checking of a namespace. Returns checked AST if successful,
throws
   an exception if something is wrong.

   - nname is used just for error output
   - forbidden_files are files that if opened would cause a circular
dependency
   - files are a list of filename-namespace associations
   - curr_dir is used for resolving file opens in this namespace

   Check each inner namespace, global variable, function, and then gameobj. *)

let rec check_namespace (nname, namespace) forbidden_files files curr_dir =
  let { Namespace.variables = globals; functions ; gameobjs; namespaces = _;
usings = _ } = namespace in
```

```
  (**** Checking Namespaces ****)
  (* Add the standard namespace and mark private *)
  let namespace =
    if List.mem (File.resolve_file "std.mg" curr_dir) forbidden_files
    then namespace
    else
      let namespaces = ("std", (true, Namespace.File "std.mg")) ::
namespace.Namespace.namespaces in
      { namespace with Namespace.namespaces }
  in

  report_duplicate (fun n -> "duplicate namespace " ^ nname ^ "::" ^ n)
    (List.map fst namespace.Namespace.namespaces);

  (* Check inner namespaces and populate list of files *)
  let namespace, files =
    let open Namespace in
    (* Check and update nested concrete namespaces *)
    let namespaces =
      let check_concrete = function
        | n, (p, Concrete ns) ->
          let (n, ns), _ = check_namespace (n, ns) forbidden_files files
curr_dir in
          (n, (p, Concrete ns))
        | _ as ns -> ns
      in
      List.map check_concrete namespace.namespaces
    in
    let namespace = { namespace with namespaces } in
    (* Update list of file-namespace associations by opening each file
namespace *)
    let check_file_ns (accum, ns_accum) = function
      | n, (is_private, File f) ->
        (* Convert to absolute file path and use that in the namespace
reference now *)
        let abs_f = File.resolve_file f curr_dir in
        let f_dir = File.dirname abs_f in
        let new_accum =
          if List.mem_assoc abs_f accum then accum
          else
            let { main = file_prog; files = _ } = ast_of_file abs_f f in
            let new_forbidden =
              if List.mem abs_f forbidden_files
              then failwith ("circular file dependency with " ^ f)
              else abs_f :: forbidden_files
            in
            let (_, file_prog), files =
```

```
              check_namespace ("file-" ^ f, file_prog) new_forbidden accum
f_dir
            in
            (abs_f, file_prog) :: files
        in
        new_accum, (n, (is_private, File abs_f)) :: ns_accum
      | _ as next_ns -> accum, (next_ns :: ns_accum)
    in
    let files, namespaces = List.fold_left check_file_ns (files, [])
namespaces in
    (* undo the reverse from the fold_left. should be right either way but
this has fewer changes *)
    let namespaces = List.rev namespaces in
    let namespace = { namespace with namespaces } in
    (* Verify that each alias indeed resolves to a namespace *)
    let () =
      let aliases =
        List.fold_left
          (fun a (_, (_, x)) -> match x with Alias chain -> chain :: a | _ ->
a) []
          namespace.namespaces
      in
      let check_resolve chain =
        ignore (namespace_of_chain namespace [] files true chain
(String.concat "::" chain))
      in
      List.iter check_resolve aliases
    in
    namespace, files
  in
  let namespace_of_chain chain =
    namespace_of_chain namespace [] files true chain (String.concat "::"
chain)
  in

  (**** Checking Global Variables ****)

  let () =
    let global_binds = List.map fst globals in
    List.iter (check_not_void (fun n -> "illegal void global " ^ nname ^ "::"
^ n)) global_binds;
    report_duplicate (fun n -> "duplicate global " ^ nname ^ "::" ^ n)
(List.map fst global_binds)
  in

  let check_valid_const_assign v =
    let str = string_of_global_vdecl v in
    let (_, t), e = v in
```

```
    let rec const_expr = function
      | Literal _ -> Int
      | BoolLit _ -> Bool
      | FloatLit _ -> Float
      (* | StringLit _ -> String *)   (* implementation isn't that trivial...
*)
      | ArrayLit l ->
        (match l with
         | [] -> failwith ("illegal empty array in " ^ str)
         | hd :: tl ->
           let lt = const_expr hd in
           List.iter (fun l ->
               let rt = const_expr l in
               if rt <> lt then
                 failwith ("array literal " ^ string_of_expr e ^
                           " contains elements of unequal types "
                           ^ string_of_typ lt ^ " and " ^ string_of_typ rt))
tl;
           Arr (lt, List.length l))
      | _ -> failwith ("illegal assignment to global variable: " ^ str)
    in
    match e with
    | Noexpr -> ()
    | _ ->
      let lt = const_expr e in
      if t = lt then () else
        failwith ("literal (" ^ string_of_typ lt ^ ") and global (" ^
                  string_of_typ t ^ ") type mismatch in " ^ str)
  in
  List.iter check_valid_const_assign globals;

  (**** Checking Functions ****)

  report_duplicate (fun n -> "duplicate function " ^ nname ^ "::" ^ n)
(List.map fst functions);
  report_duplicate (fun n -> "duplicate gameobj " ^ nname ^ "::" ^ n)
(List.map fst gameobjs);

  (* TODO: stop using add_to_scope *)
  (* Check each gameobj declaration *)
  ignore (List.fold_left (add_to_scope ~loc:(nname ^ "gameobj declarations"))
            StringMap.empty gameobjs);

  let gameobj_decl (chain, s) =
    match (chain, s) with
    | _, "object" -> Gameobj.generic
    | _ ->
      try List.assoc s (namespace_of_chain chain).Namespace.gameobjs
```

```
      with Not_found -> failwith ("unrecognized game object " ^ nname ^ "::"
^ s)
  in

  (* Inheritance chain from oldest ancestor down. *)
  let inheritance_chain (chain, name) =
    let rec helper (chain, name) accum =
      let decl = gameobj_decl (chain, name) in
      (* Physical equality as a kind of 'comparing handles' *)
      (if List.memq decl accum
       then failwith ("inheritance cycle with " ^ string_of_chain (chain,
name))
       else ());
      match decl.Gameobj.parent with
      | None -> decl :: accum
      | Some (pchain, pname) ->
        helper ((chain @ pchain), pname) (decl :: accum)
    in
    helper (chain, name) []
  in

  let is_gameobj_parent p c =
    match c with
    | [], "none" -> true        (* None can be assigned to anything *)
    | _ -> List.exists ((==) (gameobj_decl p)) (inheritance_chain c)
  in

  (* Raise an exception if the given rvalue type cannot be assigned to
     the given lvalue type, or else return a new expression with conversion.
*)
  let check_assign lvaluet rvalue rvaluet err =
    match lvaluet, rvaluet with
    | Float, Int -> (Float, Conv(Float, rvalue, Int))
    | Int, Float -> (Int, Conv(Int, rvalue, Float))
    | Object p, Object c when is_gameobj_parent p c ->
      (Object p, Conv(Object p, rvalue, Object c))
    (* TODO: array conversions? e.g. int[3] to/from int[5]? int[3] to
float[3]? *)
    | _ -> if lvaluet = rvaluet then (lvaluet, rvalue) else failwith err
  in

  (* Helper for getting a set of things from a gameobj and its parents. *)
  let gscope_helper name retrieve loc =
    let gscope parent_scope decl =
      let indiv_scope =
        List.fold_left (add_to_scope ~loc:(nname ^ "::" ^ string_of_chain
name ^ " " ^ loc))
          StringMap.empty (retrieve decl)
```

```
      in
      StringMap.fold StringMap.add indiv_scope parent_scope
    in
    List.fold_left gscope StringMap.empty (inheritance_chain name)
  in

  let gameobj_functions name =
    gscope_helper name (fun x -> x.Gameobj.methods) "method declarations"
  in

  let gameobj_scope name =
    gscope_helper name (fun x -> x.Gameobj.members) "members"
  in

  (* Object types from inner namespaces need to include the outer calling
     namespace to remain valid references. Any expr that could resolve to a
     type defined in an inner namespace needs this added. *)
  let add_typ_ns chain = function
    | Object (c, n) -> Object (chain @ c, n)
    | _ as t -> t
  in
  (* Check that the expression can indeed be assigned to *)
  let rec check_lvalue loc = function
    | Id([], "this") -> failwith ("'this' cannot be assigned in '" ^ loc ^
"'")
    | Id([], "super") -> failwith ("'super' cannot be assigned in '" ^ loc ^
"'")
    | Subscript(arr, _) -> check_lvalue loc arr (* subscr is lvalue iff arr
is *)
    | Id _ | Member _ | Assign _ | Asnop _ | Idop _ -> ()
    | _ as e -> failwith ("lvalue " ^ string_of_expr e ^ " expected in " ^
loc)
  in
  (* Return the type of an expression and the new expression or throw an
exception *)
  let rec expr scope e = match e with
    | Literal _ -> Int, e
    | BoolLit _ -> Bool, e
    | FloatLit _ -> Float, e
    | StringLit _ -> String, e
    | NoneObject -> Object([], "none"), e
    | Conv(t1, e, _) ->
      let (t2, e') = expr scope e in
      check_assign t1 e' t2 ("Cannot convert " ^ string_of_typ t2 ^ " to " ^
                             string_of_typ t1 ^ " in " ^ string_of_expr e')
    | ArrayLit l ->
      (* Check array size and equality of element types *)
```

```
      (* TODO: in LRM say that implicit conversions do not apply to array
literals *)
      (match l with
       | [] -> failwith ("illegal empty array " ^ string_of_expr e)
       | hd :: tl ->
         let (lt, hd') = expr scope hd in
         let tl' = List.map (fun l ->
             let t, e' = expr scope l in
             let _, e'' = check_assign_strict lt e' t
                 ("array literal " ^ string_of_expr e ^
                  " contains elements of unequal types "
                  ^ string_of_typ lt ^ " and " ^ string_of_typ t)
             in e'') tl
         in
         Arr (lt, List.length l), ArrayLit(hd' :: tl'))
    | Binop(e1, op, _, e2) ->
      let (t1, e1') = expr scope e1 and (t2, e2') = expr scope e2 in
      let is_object t = match t with Object _ -> true | _ -> false in
      let err = "illegal binary operator " ^ string_of_typ t1 ^ " " ^
string_of_op op ^ " " ^
                string_of_typ t2 ^ " in " ^ string_of_expr e
      in
      (match op with
       | Add | Sub | Mult | Div | Modulo when t1 = Int && t2 = Int -> Int,
Binop(e1', op, Int, e2')
       | Add | Sub | Mult | Div | Modulo when t1 = Float && t2 = Float ->
Float, Binop(e1', op, Float, e2')
       | Add | Sub | Mult | Div | Modulo when t1 = Float && t2 = Int ->
         let (_, e2'') = check_assign t1 e2' t2 err
         in Float, Binop(e1', op, Float, e2'')
       | Add | Sub | Mult | Div | Modulo when t1 = Int && t2 = Float ->
         let (_, e1'') = check_assign t2 e1' t1 err
         in Float, Binop(e1'', op, Float, e2')
       (* TODO: mention in LRM that we are not converting bools *)
       (* TODO: string, obj equality *)
       | Equal | Neq when t1 = t2 && (t1 = Float || t1 = Int) -> Bool,
Binop(e1', op, t1, e2')
       | Equal | Neq when is_object t1 && is_object t2 ->
         (match t1, t2 with
          | Object o1, Object o2 ->
            if is_gameobj_parent o1 o2
            then let (_, e2') = check_assign t1 e2' t2 err in Bool, Binop(e1',
op, t1, e2')
            else let (_, e1') = check_assign t2 e1' t1 err in Bool, Binop(e1',
op, t2, e2')
          | _ -> assert false)
       | Equal | Neq when t1 = Float && t2 = Int ->
         let (_, e2'') = check_assign t1 e2' t2 err
```

```
        in Bool, Binop(e1', op, Float, e2'')
      | Equal | Neq when t1 = Int && t2 = Float ->
        let (_, e1'') = check_assign t2 e1' t1 err
        in Bool, Binop(e1'', op, Float, e2')
      | Less | Leq | Greater | Geq when t1 = Int && t2 = Int -> Bool,
Binop(e1', op, Int, e2')
      | Less | Leq | Greater | Geq when t1 = Float && t2 = Float -> Bool,
Binop(e1', op, Float, e2')
      | Less | Leq | Greater | Geq when t1 = Float && t2 = Int ->
        let (_, e2'') = check_assign t1 e2' t2 err
        in Bool, Binop(e1', op, Float, e2'')
      | Less | Leq | Greater | Geq when t1 = Int && t2 = Float ->
        let (_, e1'') = check_assign t2 e1' t1 err
        in Bool, Binop(e1'', op, Float, e2')
      | And | Or when t1 = Bool && t2 = Bool -> Bool, Binop(e1', op, Bool,
e2')
      | _ -> failwith err)
    | Unop(op, _, ex) -> let (t, ex') = expr scope ex in
      (match op, t with
       | Neg, Int | Neg, Float -> t, Unop(op, t, ex')
       | Not, Bool -> Bool, Unop(op, Bool, ex')
       | _ -> failwith ("illegal unary operator " ^ string_of_uop op ^
                        string_of_typ t ^ " in " ^ string_of_expr e))
    | Idop (opid, _, e1) ->
      check_lvalue (string_of_expr e) e1;
      let (t, e1') = expr scope e1 in
      (match t with
       | Int -> Int, Idop(opid, Int, e1')
       | Float -> Float, Idop(opid, Float, e1')
       | _ -> failwith ("illegal Increment/Decrement operator " ^
                        string_of_typ t ^ " " ^ string_of_idop opid))
    | Noexpr -> Void, e
    | Assign(l, r) ->
      check_lvalue (string_of_expr e) l;
      let (lt, l') = expr scope l and (rt, r') = expr scope r in
      let (t'', r'') = check_assign lt r' rt ("illegal assignment " ^
string_of_typ lt ^
                                              " = " ^ string_of_typ rt ^ " in
" ^
                                              string_of_expr e)
      in t'', Assign(l', r'')
    | Asnop(e1, opasn, _, e2) ->
      check_lvalue (string_of_expr e1) e1;
      let (t1, e1) = expr scope e1 and (t2, e2) = expr scope e2 in
      let err =
        "illegal assign operator " ^
        string_of_typ t1 ^ " " ^ string_of_asnop opasn ^ " " ^
        string_of_typ t2 ^ " in " ^ string_of_expr e
```

```
      in
      (match t1, t2 with
       | Int, Int | Float, Float -> t1, Asnop(e1, opasn, t1, e2)
       | Float, Int | Int, Float ->
         let t2, e2 = check_assign t1 e2 t2 err in
         assert (t1 = t2); t1, Asnop(e1, opasn, t1, e2)
       | _ -> failwith err)
    | Id (chain, name) ->
      let t =
        try match chain with
          | [] -> let vscope, _ = scope in StringMap.find name vscope
          | _ ->
            let var_decls = List.map fst (namespace_of_chain
chain).Namespace.variables in
            add_typ_ns chain (List.assoc name var_decls)
        with Not_found -> failwith ("undeclared identifier " ^
(string_of_chain (chain, name)))
      in t, e
    | Subscript(arr, ind) ->
      let ind' =
        match expr scope ind with
        | Int, ind' -> ind'
        | _ -> failwith ("expected integer index " ^ string_of_expr ind ^
                         " in " ^ string_of_expr e)
      in
      let elem_type, arr' =
        match expr scope arr with
        | Arr (t, _), arr' -> t, arr'
        | _ -> failwith ("expected array " ^ string_of_expr arr ^
                         " in " ^ string_of_expr e)
      in
      elem_type, Subscript(arr', ind')
    | Member(e, _, name) ->
      (match expr scope e with
       | Object s, e' ->
         let t =
           try StringMap.find name (gameobj_scope s)
           with Not_found ->
             failwith ("undefined member " ^ name ^ " in " ^
                       string_of_expr e ^ " of type " ^ string_of_chain s)
         in
         let ochain, _ = s in add_typ_ns ochain t, Member(e', s, name)
       | _ -> failwith ("cannot get member of non-object " ^ (string_of_expr
e)))
    | Call((chain, fname), actuals) as call ->
      let fd =
        try match chain with
          | [] -> let _, fscope = scope in StringMap.find fname fscope
```

```
          | _ -> List.assoc fname (namespace_of_chain
chain).Namespace.functions
        with Not_found -> failwith ("unrecognized function " ^
(string_of_chain (chain, fname)))
      in
      let actuals' = check_call_actuals chain (string_of_expr call) actuals
scope fd.Func.formals in
      add_typ_ns chain fd.Func.typ, Call((chain, fname), actuals')
    | MemberCall(e, _, fname, actuals) as call ->
      let fd, (ochain, oname), e' = match expr scope e with
        | Object s, e' ->
          (try StringMap.find fname (gameobj_functions s), s, e'
           with Not_found ->
             failwith ("undefined member " ^ fname ^ " in " ^
                       string_of_expr e ^ " of type " ^ string_of_chain s))
        | _ -> failwith ("cannot get member of non-object " ^ (string_of_expr
e))
      in
      let actuals' = check_call_actuals ochain (string_of_expr call) actuals
scope fd.Func.formals in
      add_typ_ns ochain fd.Func.typ, MemberCall(e', (ochain, oname), fname,
actuals')
    | Create((ochain, oname), actuals) ->
      let formals =
        (* Find the create event formals, recursing up the inheritance chain.
*)
        let get_formals formals decl =
          if List.mem_assoc "create" decl.Gameobj.events
          then (List.assoc "create" decl.Gameobj.events).Func.formals
          else formals
        in
        List.fold_left get_formals [] (inheritance_chain (ochain, oname))
      in
      let actuals' = check_call_actuals ochain (string_of_expr e) actuals
scope formals in
      Object(ochain, oname), Create((ochain, oname), actuals')
    | Destroy e ->
      (match expr scope e with
       | Object _, e' -> Void, Destroy e'
       | _ -> failwith ("cannot destroy non-object"))
    | Delete e ->
      (match expr scope e with
       | Object _, e' -> Void, Delete e'
       | _ -> failwith ("cannot delete non-object"))
  and check_call_actuals chain loc actuals scope formals =
    if List.length actuals != List.length formals then
      failwith ("expecting " ^ string_of_int (List.length formals) ^
                " arguments in " ^ loc)
```

```
    else
      let formals = List.map (fun (n, t) -> n, add_typ_ns chain t) formals in
      List.map2
        (fun (_, ft) ex -> let (et, ex') = expr scope ex in
          let (_, ex'') = check_assign ft ex' et
              ("illegal actual argument found " ^ string_of_typ et ^
               " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr ex)
in ex'')
        formals actuals
  in

  let check_bool_expr scope e = let (t, e') = expr scope e in if t != Bool
    then failwith ("expected Boolean expression in " ^ string_of_expr e)
    else e' in

  let rec check_block ~scope ~name ~return block =
    (* TODO: say in LRM it's okay to duplicate locals/formals now *)
    (* TODO: clarify in LRM where STATIC scope starts and ends for decls *)

    (* Check duplicate locals *)
    List.fold_left
      (fun a n -> match n with Decl (n, _) -> n :: a | _ -> a) [] block
    |> report_duplicate
      (fun n -> "duplicate local '" ^ n ^ "' in single block of " ^ name);

    (* Verify a statement or throw an exception *)
    let rec stmt scope = function
      | Decl d as s ->
        let vscope, fscope = scope in
        (* Make sure it's actually an object *)
        (match d with | _, Object o -> ignore(gameobj_decl o) | _ -> ());
        check_not_void (fun n -> "illegal void local " ^ n ^ " in " ^ name) d;
        s, (add_to_scope ~loc:name vscope d, fscope)
      | Vdef((id, t), e) ->
        let (et, e') = expr scope e in
        let (_, e'') = check_assign t e' et ("illegal assignment " ^
string_of_typ t ^
                                             " = " ^ string_of_typ et ^ " in
" ^
                                             string_of_expr e) in
        let vscope, fscope = scope in
        check_not_void (fun n -> "illegal void local " ^ n ^ " in " ^ name)
(id, t);
        Vdef((id, t), e''), (add_to_scope ~loc:name vscope (id, t), fscope)
      | Break -> Break, scope
      | Block b -> Block (check_block b ~scope ~name ~return), scope
      | Expr e -> let (_, e') = expr scope e in Expr e', scope
```

```
      | Return e ->             (* TODO in LRM say stuff can follow returns
*)
        let t, e' = expr scope e in
        let (_, e'') = check_assign return e' t ("return gives " ^
string_of_typ t ^ " expected " ^
                                                 string_of_typ return ^ " in
" ^ string_of_expr e) in
        Return e'', scope
      | If(p, b1, b2) ->
        let (b1', _), (b2', _) = stmt scope b1, stmt scope b2 in
        If (check_bool_expr scope p, b1', b2'), scope
      | For(s1, e2, e3, st) ->
        let s1', scope' = stmt scope s1 in let e2' = check_bool_expr scope' e2
in
        let (_, e3') = expr scope' e3 in let st', _ = stmt scope' st in
        For (s1', e2', e3', st'), scope
      | While(p, s) ->
        let p' = check_bool_expr scope p in let s', _ = stmt scope s in
        While(p', s'), scope
      | Foreach((id, typ), s) ->
        match typ with
        | Object obj_t ->
          ignore(gameobj_decl obj_t);
          let vscope, fscope = scope in
          let s', _ = stmt (StringMap.add id (Object(obj_t)) vscope, fscope)
s in
          Foreach((id, typ), s'), scope
        | _ -> failwith ("can only use foreach loop on objects in " ^
(string_of_vdecl (id, typ)))
    in

    let _, block' =
      List.fold_left
        (fun (sym, accum) s -> let s', sym' = stmt sym s in (sym', s' ::
accum))
        (scope, []) block
    in
    List.rev block'
  in
  let check_block ~scope ~name ~return block =
    List.iter
      (function Break ->
         failwith ("cannot break in top level block of " ^ name) | _ -> ())
      block;
    check_block ~scope ~name ~return block
  in

  let check_function ~scope ~objname (name, func) =
```

```
    let open Func in
    let loc = match objname with
      | Some objname -> nname ^ "::" ^ objname ^ "::" ^ name
      | None -> nname ^ "::" ^ name
    in
    List.iter
      (check_not_void (fun n -> "illegal void formal " ^ n ^ " in " ^ loc))
      func.formals;

    let scope =
      let vscope, fscope = scope in
      List.fold_left (add_to_scope ~loc:("formals of " ^ loc))
        vscope func.formals, fscope
    in

    (* formals can have the same name as locals. is this okay? think about
these
       edge cases *)
    report_duplicate
      (fun n -> "duplicate formal " ^ n ^ " in " ^ loc)
      (List.map fst func.formals);

    let block =
      match func.block with
      | Some block ->
        Some (check_block ~scope ~name:loc ~return:func.typ block)
      | None -> None
    in
    name, { func with block = block }
  in

  let check_gameobj ~scope (name, obj) =
    let open Gameobj in
    report_duplicate
      (fun n -> "duplicate members " ^ n ^ " in " ^ name)
      (List.map fst obj.members);

    report_duplicate
      (fun n -> "duplicate methods " ^ n ^ " in " ^ name)
      (List.map fst obj.methods);

    report_duplicate
      (fun n -> "duplicate events " ^ n ^ " in " ^ name)
      (List.map fst obj.events);

    let () =
      (* Precautionary checks that shouldn't happen b/c of parser *)
      let valid_events = ["create"; "step"; "draw"; "destroy"] in
```

```
      let check_event (name, ev) =
        match (name, ev.Func.formals, ev.Func.typ) with
        | name, _, Void when name = "create" -> ()
        | name, args, Void when List.mem name valid_events && args = [] -> ()
        | _ -> assert false
      in
      List.iter check_event obj.events
    in

    (* Checks that no members are named 'this', and that the parent, if set,
       exists. Also detects inheritance cycles. *)
    ignore (gameobj_scope ([], name));

    (* For events, also add super() *)
    let super_scope event (vscope, fscope) =
      match obj.parent with
      | None | Some (_, "object") -> vscope, fscope
      | Some pname ->
        (* Find the event, recursing up the inheritance chain. *)
        let get_event formals decl =
          if List.mem_assoc event decl.Gameobj.events
          then List.assoc event decl.Gameobj.events
          else formals
        in
        let empty_event = Func.make Void [] (Some "object") [] in
        let event = List.fold_left get_event empty_event (inheritance_chain
pname) in
        vscope, StringMap.add "super" event fscope
    in

    let scope =
      let initial_vscope, initial_fscope = scope in
      (* Add "this" and gameobj members to scope *)
      let vscope =
        initial_vscope
        |> StringMap.fold StringMap.add (gameobj_scope ([], name))
        |> StringMap.add "this" (Object([], name))
      in
      let fscope =
        initial_fscope
        |> StringMap.fold StringMap.add (gameobj_functions ([], name))
      in
      (vscope, fscope)
    in

    let check_obj_fn (fname, func) =
      match func.Func.block with
      | Some _ ->
```

```
        let scope =
          if List.mem fname ["create"; "step"; "draw"; "destroy"]
          then super_scope fname scope else scope
        in
        check_function ~scope ~objname:(Some name) (fname, func)
      | _ -> failwith ("illegal extern function " ^ nname ^ "::" ^ name ^
"::" ^ fname)
    in
    let methods' = List.map check_obj_fn obj.methods in
    let events' = List.map check_obj_fn obj.events in
    make name (obj.members, methods', events') obj.parent
  in

  let usings =
    List.iter (fun (_priv, ns) -> ignore (namespace_of_chain ns))
namespace.Namespace.usings;
    namespace.Namespace.usings
  in
  let scope = namespace_scope namespace_of_chain nname in
  let functions = List.map (check_function ~objname:None ~scope) functions in
  let gameobjs = List.map (check_gameobj ~scope) gameobjs in
  let namespaces = namespace.Namespace.namespaces in

  (nname, { Namespace.variables = globals; functions; gameobjs; namespaces;
usings }), files
;;

let check_program program =
  let (_, main), files = check_namespace ("", program.main) [] program.files
(Sys.getcwd ()) in

  if not (List.mem_assoc "main" main.Namespace.gameobjs)
  then failwith "main game object must be defined"
  else { main; files }
```

codegen.ml

```
module L = Llvm
module A = Ast
module B = Llast

module StringMap = Map.Make(String)

module Gconfig = struct
  type scoped_block_builder =
    (L.llbuilder -> (B.L.llvalue * A.Var.t) StringMap.t * B.func StringMap.t
-> L.llbuilder)
  type t = {
```

```
    obj: string;
    postwork: scoped_block_builder option; (* Unused now. Used to be for
removing nodes. *)
    super: string option;
  }
end

(* Helper function for assigning struct values. *)
let build_struct_assign str values builder =
  let assign (llv, ind) value =
    match value with
    | Some v -> (L.build_insertvalue llv v ind "" builder, ind+1)
    | None -> (llv, ind+1)
  in
  let (ret, _) = Array.fold_left assign (str, 0) values in ret

(* Helper function for assigning struct values given a pointer. *)
let build_struct_ptr_assign ptr values builder =
  let assign ind value =
    match value with
    | Some v ->
      ignore (L.build_store v (L.build_struct_gep ptr ind "" builder) builder)
    | None -> ()
  in Array.iteri assign values

(* Given value v that's position offset in aggregate type container_t, get a
pointer
   to the container of v, casted to cast. *)
let build_container_of container_t ?(cast=container_t) offset v name context
builder =
  let null = L.const_null (L.pointer_type container_t) in
  let offset = L.build_struct_gep null offset "offset" builder in
  let offsetint = L.build_ptrtoint offset (L.i64_type context) "offsetint"
builder in
  let intptr = L.build_ptrtoint v (L.i64_type context) "intptr" builder in
  let intnew = L.build_sub intptr offsetint "intnew" builder in
  (* build_print "%d %d %d" [offsetint; intptr; intnew] builder; *)
  L.build_inttoptr intnew (L.pointer_type cast) name builder

let build_if ~pred ~then_ ?(else_ = (fun b _ -> b)) context builder
the_function =
  let bool_val = pred builder the_function in
  let merge_bb = L.append_block context "merge" the_function in

  let then_bb = L.append_block context "then" the_function in
  let then_builder = then_ (L.builder_at_end context then_bb) the_function in
  ignore (L.build_br merge_bb then_builder);
```

```
  let else_bb = L.append_block context "else" the_function in
  let else_builder = else_ (L.builder_at_end context else_bb) the_function in
  ignore (L.build_br merge_bb else_builder);

  ignore (L.build_cond_br bool_val then_bb else_bb builder);
  L.builder_at_end context merge_bb

(* Builds a while LLVM construct given a predicate construct (returning a bool
   llvalue and possibly some information to body) and a body construct. *)
let build_while ~pred ~body context builder the_function =
  let pred_bb = L.append_block context "while" the_function in
  let body_bb = L.append_block context "while_body" the_function in
  let merge_bb = L.append_block context "merge" the_function in
  ignore (L.build_br pred_bb builder);

  let pred_builder = L.builder_at_end context pred_bb in
  let bool_val, transfer = pred pred_builder the_function in

  let body_builder =
    body (L.builder_at_end context body_bb) merge_bb the_function transfer
  in
  ignore (L.build_br pred_bb body_builder);

  ignore (L.build_cond_br bool_val body_bb merge_bb pred_builder);
  L.builder_at_end context merge_bb

let rec namespace_of_chain llfiles top chain =
  let search ns n = match StringMap.find n ns.B.namespaces with
    | B.Concrete c -> c
    | B.Alias chain -> namespace_of_chain llfiles ns chain
    | B.File f -> StringMap.find f llfiles
  in
  List.fold_left search top chain

let translate the_program files =
  let context = L.global_context () in
  let the_module = L.create_module context "MicroC"
  and i64_t    = L.i64_type    context
  and i32_t    = L.i32_type    context
  and i8_t     = L.i8_type     context
  and i1_t     = L.i1_type     context
  and float_t  = L.double_type context (* TODO: fix LRM to say double
precision *)
  and sprite_t = L.pointer_type (L.named_struct_type context "sfSprite")
  and sound_t  = L.pointer_type (L.named_struct_type context "sfSound")
  and void_t   = L.void_type   context in

  (* Declare types for each object type *)
```

```
  (* Define generic types for linked lists and game objects *)
  let node_t = L.named_struct_type context "node" in
  let nodeptr_t = L.pointer_type node_t in
  L.struct_set_body node_t [|nodeptr_t; nodeptr_t; i1_t|] false; (* 0 if node
is not assoc. with object *)

  let gameobj_t = L.named_struct_type context "gameobj" in
  let objptr_t = L.pointer_type gameobj_t in
  let objref_t = L.named_struct_type context "objref" in
  L.struct_set_body objref_t [|i64_t; objptr_t|] false;

  let eventptr_t = L.pointer_type (L.function_type void_t [|objref_t|]) in
  let gameobj_vtable = L.struct_type context [|eventptr_t; eventptr_t;
eventptr_t; eventptr_t|] in
  let vtableptr_t = L.pointer_type gameobj_vtable in
  L.struct_set_body gameobj_t [|vtableptr_t; node_t; i64_t|] false;

  (* Define linked list heads for the generic gameobj *)
  let make_node_end name =
    let h = L.declare_global node_t ("node." ^ name ^ ".head") the_module in
    L.set_initializer (L.const_named_struct node_t [|h; h; L.const_int i1_t
0|]) h; h
  in
  let gameobj_head = make_node_end "gameobj" in

  let global_objid = L.define_global "last_objid" (L.const_int i64_t 0)
the_module in

  let rec ltype_of_typ = function
    | A.Int -> i32_t
    | A.Bool -> i1_t
    | A.Float -> float_t
    | A.Arr (typ, len) -> L.array_type (ltype_of_typ typ) len
    | A.String -> L.pointer_type i8_t
    | A.Sprite -> sprite_t
    | A.Sound -> sound_t
    | A.Object _ -> objref_t
    | A.Void -> void_t
  in

  (* Define list_add(new, head), which puts new to the end of the list
     marked by head. *)
  let list_add_func =
    let f =
      let t = L.function_type void_t [|nodeptr_t; nodeptr_t|] in
      L.define_function "list_add" t the_module
    in
    let builder = L.builder_at_end context (L.entry_block f) in
```

```
    let node, head = L.param f 0, L.param f 1 in
    let following_prev_ptr = L.build_struct_gep head 0 "prev_ptr" builder in
    let fprev = L.build_load following_prev_ptr "prev" builder in
    let preceding_next_ptr = L.build_struct_gep fprev 1 "next_ptr" builder in
    (* let pnext = L.build_load preceding_next_ptr "next" builder in *)
    let pnext = head in
    ignore (L.build_store node following_prev_ptr builder);
    ignore (L.build_store node preceding_next_ptr builder);
    build_struct_ptr_assign node [|Some fprev; Some pnext|] builder;
    ignore (L.build_ret_void builder);
    f
  in

  (* Removes a node from a linked list. Does not free. *)
  let list_rem_func =
    let f =
      let t = L.function_type void_t [|nodeptr_t|] in
      L.define_function "list_rem" t the_module
    in
    let builder = L.builder_at_end context (L.entry_block f) in
    let node = L.param f 0 in
    let prev_ptr = L.build_struct_gep node 0 "prev_ptr" builder in
    let prev = L.build_load prev_ptr "prev" builder in
    let next_ptr = L.build_struct_gep node 1 "next_ptr" builder in
    let next = L.build_load next_ptr "next" builder in
    let next_prev = L.build_struct_gep next 0 "next_prev" builder in
    let prev_next = L.build_struct_gep prev 1 "prev_next" builder in
    (* TODO: currently reconnects only if they originally connected to you. *)
    (* If the linked list system were really robust we wouldn't need this...
it
       causes some newly created objects to skip a step. *)
    (* Something really insidious shows up in Tetris without this guard,
though,
       and I don't have the time to figure it out. *)
    let builder =
      build_if context builder f
        ~pred:(fun builder _ ->
            let next_prev_val = L.build_load next_prev "" builder in
            L.build_icmp L.Icmp.Eq next_prev_val node "cmp" builder)
        ~then_:(fun builder _ -> ignore (L.build_store prev next_prev
builder); builder)
    in
    let builder =
      build_if context builder f
        ~pred:(fun builder _ ->
            let prev_next_val = L.build_load prev_next "" builder in
            L.build_icmp L.Icmp.Eq prev_next_val node "cmp" builder)
```

```
        ~then_:(fun builder _ -> ignore (L.build_store next prev_next
builder); builder)
    in
    ignore (L.build_ret_void builder); f
  in

  let delete_function ltyp =
    let name = "delete_" ^ L.string_of_lltype ltyp in
    match L.lookup_function name the_module with
    | Some f -> f
    | None ->
      let f =
        let t = L.function_type void_t [|objref_t|] in
        L.define_function name t the_module
      in
      let builder = L.builder_at_end context (L.entry_block f) in
      let llobj =
        let objptr = L.build_extractvalue (L.param f 0) 1 "objptr" builder in
        L.build_bitcast objptr (L.pointer_type ltyp) "" builder
      in
      let rec helper llobj =
        (* print_endline (L.string_of_lltype (L.type_of llobj)); *)
        (* Disconnect particular object neighbours' node links. *)
        let llobjnode = L.build_struct_gep llobj 1 "objnode" builder in
        ignore (L.build_call list_rem_func [|llobjnode|] "" builder);
        (* Call parent destroy *)
        let next = L.build_struct_gep llobj 0 "" builder in
        if L.type_of llobj = (L.pointer_type gameobj_t)
        then ignore (L.build_ret_void builder)
        else helper next
      in
      helper llobj; f
  in

  (* Make a B.gameobj for the generic game object *)
  let empty_method =
    let typ = L.function_type void_t [|objref_t|] in
    let value = L.define_function "_empty_fn" typ the_module in
    let return = A.Void in
    ignore (L.build_ret_void (L.builder_at_end context (L.entry_block
value)));
    { B.value; typ; return; gameobj = (Some "object") }
  in
  let gameobj_struct =
    let vtable_gen =
      let fn = empty_method.B.value in
      let del_fn = delete_function gameobj_t in
```

```
      L.define_global ("object.vtable") (L.const_struct context [|fn; fn; fn;
del_fn|]) the_module
    in
    let events =
      ["create"; "step"; "destroy"; "draw"]
      |> List.fold_left (fun m x -> StringMap.add x empty_method m)
StringMap.empty
    in
    { B.gtyp = gameobj_t;
      head = gameobj_head;
      methods = StringMap.empty;
      events; vtable = vtable_gen;
      semant = A.Gameobj.generic }
  in

  (* 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 build_print fmt elems builder = *)
  (*   ignore (L.build_call printf_func *)
  (*             (Array.of_list (L.build_global_stringptr (fmt ^ "\n") ""
builder *)
  (*                             :: elems)) "" builder) *)
  (* in *)
  let build_print _ _ _ = () in

  let build_printstr str builder =
    build_print "%s" [L.build_global_stringptr str "" builder] builder
  in


  let build_obj_cmp ~eq e1 e2 name builder =
    let lid = L.build_extractvalue e1 0 (name ^ "_lid") builder in
    let rid = L.build_extractvalue e2 0 (name ^ "_rid") builder in
    L.build_icmp (if eq then L.Icmp.Eq else L.Icmp.Ne) lid rid name builder
  in

  let build_delete ~destroy objref builder =
    (* Deletion is lazy in that it involves just removing the
       object-specific link from an node's neighbours (i.e. used for foreach)
       and setting id to 0. When the main event loop encounters an object with
       id 0, it looks ahead to the next node and deletes the current. Other
       loops ignore 0-id objects.

       This way, a kind of induction can show that there's always a forward
       path from such an object to the head of the list (its neighbours don't
       point to it, but it still points to what it thinks are its neighbours
       have a forward path). So a loop like a foreach that might end up on a
```

```
       dead object still behaves well. The main event loop is guaranteed to
       run outside of any other loops, so it's safe to actually delete the
       nodes there. *)
    (* TODO: describe setup and effects in LRM. destroyed objs are immediately
       invisible to loops but refs to them still work until at least end of
       event. *)
    let objptr = L.build_extractvalue objref 1 "objptr" builder in
    (* Call its destroy event *)
    let destroy_event, delete_event =
      (* Assuming vtable is at front *)
      let tbl =
        L.build_load (L.build_bitcast objptr (L.pointer_type vtableptr_t) ""
builder) "tbl" builder
      in
      L.build_load (L.build_struct_gep tbl 1 "eventptr" builder) "event"
builder,
      L.build_load (L.build_struct_gep tbl 3 "eventptr" builder) "event"
builder
    in
    (if not destroy then ()
     else ignore (L.build_call destroy_event [|objref|] "" builder));
    ignore (L.build_call delete_event [|objref|] "" builder);
    (* Mark its id as 0 *)
    let idptr = L.build_struct_gep objptr 2 "id" builder in
    ignore (L.build_store (L.const_int i64_t 0) idptr builder);
    builder
  in

  let build_node_loop builder the_function ~head ~body =
    (* Keep track of curr and next in case curr is modified when body is run.
*)
    (* If something's added right in front of the iterator, it'll be skipped.
To
       ensure adding right behind head is always safe, an empty tail node is
       added to ensure a space between the iterator and the end. *)
    let tail = L.build_alloca node_t "tail" builder in
    let marker = L.build_struct_gep tail 2 "marker" builder in
    ignore (L.build_store (L.const_int i1_t 0) marker builder); (* Set marker
to 0 to indicate not an object *)
    let curr_ptr = L.build_alloca nodeptr_t "curr_ptr" builder in
    let next_ptr = L.build_alloca nodeptr_t "next_ptr" builder in
    ignore (L.build_call list_add_func [|tail; head|] "" builder);
    ignore (L.build_store head curr_ptr builder);
    build_printstr "loop" builder;

    let next = L.build_load (L.build_struct_gep head 1 "" builder) "" builder
in
    ignore (L.build_store next next_ptr builder);
```

```
    let check_head builder _ =
      (* Move forward one *)
      let curr = L.build_load next_ptr "curr" builder in
      let next =
        L.build_load (L.build_struct_gep curr 1 "" builder) "next" builder
      in

      let curr_int = L.build_ptrtoint curr i64_t "" builder in
      let next_int = L.build_ptrtoint next i64_t "" builder in
      let head_int = L.build_ptrtoint head i64_t "" builder in
      let tail_int = L.build_ptrtoint tail i64_t "" builder in
      build_print "%d %d %d %d" [curr_int; next_int; head_int; tail_int]
builder;

      ignore (L.build_store curr curr_ptr builder);
      ignore (L.build_store next next_ptr builder);
      L.build_icmp L.Icmp.Ne curr head "cont" builder, curr
    in

    let body builder break_bb _ curr =
      let handle_tail builder _ =
        let is_my_tail builder _ = L.build_icmp L.Icmp.Eq curr tail "cont"
builder in
        let move_tail builder _ =
          ignore (L.build_call list_rem_func [|tail|] "" builder);
          ignore (L.build_call list_add_func [|tail; head|] "" builder);
          builder
        in
        build_if context builder the_function ~pred:is_my_tail
~then_:move_tail
      in

      let is_object builder _ =
        let marker = L.build_struct_gep curr 2 "markerptr" builder in
        let marker_value = L.build_load marker "marker" builder in
        L.build_icmp L.Icmp.Eq marker_value (L.const_int i1_t 1) "cont"
builder
      in
      let body builder _ = body builder break_bb curr in
      build_if context builder the_function ~pred:is_object ~then_:body
~else_:handle_tail
    in
    let builder = build_while context builder the_function ~pred:check_head
~body in
    ignore (L.build_call list_rem_func [|tail|] "" builder);
    build_printstr "loop end" builder; builder
  in
```

```
  (* TODO: Should really be an association list for vtable positioning *)
  let fn_decls functions obj to_llname =
    let open A.Func in
    let function_decl m (name, func) =
      let formals =
        match obj with
        | Some o -> ("this", A.Object([], o)) :: func.formals
        | None -> func.formals
      in
      let formal_types = Array.of_list (List.map (fun (_, t) -> ltype_of_typ
t) formals) in
      let ftype = L.function_type (ltype_of_typ func.typ) formal_types in
      let d_function name =
        match func.block with
        | Some _ -> L.define_function (to_llname name)
        | None -> L.declare_function name
      in
      let func = { B.value = d_function name ftype the_module;
                   typ = ftype; return = func.typ;
                   gameobj = func.gameobj } in
      StringMap.add name func m
    in
    List.fold_left function_decl StringMap.empty functions
  in

  let rec const_expr = function
    | A.Literal i -> L.const_int i32_t i
    | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
    | A.FloatLit f -> L.const_float float_t f
    | A.ArrayLit l ->
      let lll = Array.of_list (List.map const_expr l) in
      let typ = L.type_of (Array.get lll 0) in
      L.const_array typ lll
    | _ -> assert false
  in

  let gameobj_decl llfiles top (chain, oname) =
    match (chain, oname) with
    | _, "object" -> gameobj_struct
    | _ -> StringMap.find oname (namespace_of_chain llfiles top
chain).B.gameobjs
  in

  let rec define_llns (nname, the_namespace) llfiles =
    let { A.Namespace.variables = globals; usings;
          functions ; gameobjs ; namespaces } = the_namespace in
    let llnamespaces, llfiles =
```

```
      let open A.Namespace in
      let add_ns (m, llfiles) (n, (_private, ns)) = match ns with
        | Concrete ns ->
          let inner_llns, new_llfiles = define_llns (nname ^ "::" ^ n, ns)
llfiles in
          StringMap.add n (B.Concrete inner_llns) m, new_llfiles
        | Alias chain -> StringMap.add n (B.Alias chain) m, llfiles
        | File f      ->
          let new_llfiles =
            if StringMap.mem f llfiles then llfiles
            else
              let file_ns = List.assoc f files in
              let new_ns, new_llfiles = define_llns (":file-" ^ f, file_ns)
llfiles in
              StringMap.add f new_ns new_llfiles
          in
          StringMap.add n (B.File f) m, new_llfiles
      in
      List.fold_left add_ns (StringMap.empty, llfiles) namespaces
    in

    let llnamespace =
      { B.variables = StringMap.empty; functions = StringMap.empty;
        namespaces = llnamespaces; gameobjs = StringMap.empty }
    in

    let using_decls get_decl =
      let add_llvar accum (_priv, chain) =
        let ns = namespace_of_chain llfiles llnamespace chain in
        StringMap.fold StringMap.add (get_decl ns) accum
      in
      List.fold_left add_llvar StringMap.empty usings
    in

    let llvars =
      let var m ((n, t), e) =
        let llname = "variable" ^ nname ^ "::" ^ n in
        let init =
          match e with
          | A.Noexpr -> L.const_null (ltype_of_typ t)
          | _ -> const_expr e
        in StringMap.add n (L.define_global llname init the_module, t) m
      in
      List.fold_left var (using_decls (fun x -> x.B.variables)) globals
    in

    let llfns =
      let to_llname name = "function" ^ nname ^ "::" ^ name in
```

```
      let my_fns = fn_decls functions None to_llname in
      StringMap.fold StringMap.add my_fns (using_decls (fun x ->
x.B.functions))
    in

    let llgameobjs = using_decls (fun x -> x.B.gameobjs) in
    let llnamespace =
      { B.variables = llvars; functions = llfns;
        namespaces = llnamespaces; gameobjs = llgameobjs }
    in
    let rec add_llgameobj ns (gname, g) =
      if StringMap.mem gname ns.B.gameobjs then ns
      else
        let open A.Gameobj in
        (* Declare the struct type *)
        let to_llname pref ename =
          "object" ^ nname ^ "::" ^ gname ^ "." ^ pref ^ "." ^ ename
        in
        let gtyp = L.named_struct_type context gname in

        (* Define linked list heads for each object type *)
        let head = make_node_end ("object" ^ nname ^ "::" ^ gname) in

        (* If I have a parent, get the ll information about my parent first.
           Semant guarantees no loop. *)
        let llparent, ns = match g.A.Gameobj.parent with
          | None -> gameobj_struct, ns
          | Some name ->
            let new_ns =
              match name with
              | [], pname when pname <> "object" ->
                add_llgameobj ns (pname, List.assoc pname gameobjs)
              | _ -> ns
            in
            (gameobj_decl llfiles new_ns name), new_ns
        in

        (* Set body information *)
        let members = g.A.Gameobj.members in
        let ll_members = List.map (fun (_, typ) -> ltype_of_typ typ) members
in
        L.struct_set_body gtyp
          (Array.of_list (llparent.B.gtyp :: node_t :: ll_members)) false;

        (* Fill the event map with your events, and then events from parent *)
        let events =
          let initial = fn_decls g.events (Some gname) (to_llname "event") in
          let add_from_parent events ev =
```

```
            if StringMap.mem ev events then events
            else StringMap.add ev (StringMap.find ev llparent.B.events) events
          in
          List.fold_left add_from_parent initial ["create"; "step";
"destroy"; "draw"]
        in
        (* Then fill the vtable *)
        let vtable =
          let fns =
            ["step"; "destroy"; "draw"]
            |> List.map (fun event -> (StringMap.find event events).B.value)
          in
          let fns = fns @ [delete_function gtyp] in (* Add a function for
removing nodes *)
          L.define_global (gname ^ ".vtable")
            (L.const_struct context (Array.of_list fns)) the_module
        in

        let gameobj =
          { B.gtyp; head; vtable; events; semant = g;
            methods = fn_decls g.methods (Some gname) (to_llname "function") }
        in
        { ns with B.gameobjs = StringMap.add gname gameobj ns.B.gameobjs }
    in
    List.fold_left add_llgameobj llnamespace gameobjs, llfiles
  in

  let llprogram, llfiles = define_llns ("", the_program) StringMap.empty in

  let rec define_ns_contents llns (nname, the_namespace) =
    let { A.Namespace.variables = _; usings = _;
          functions ; gameobjs ; namespaces } = the_namespace in
    let () = (* Recursively check inner namespaces *)
      let check_inner_ns (nname, (_, ns)) = match ns with
        | A.Namespace.Concrete c ->
          (match StringMap.find nname llns.B.namespaces with
           | B.Concrete llc -> define_ns_contents llc (nname, c)
           | _ -> assert false)
        | _ -> ()
      in
      List.iter check_inner_ns namespaces
    in
    let namespace_of_chain = namespace_of_chain llfiles llns in
    let gameobj_decl = gameobj_decl llfiles llns in
    (* Inheritance chain from oldest ancestor down. Semant guarantees no
loop. *)
    let inheritance_chain (chain, name) =
      let rec helper (chain, name) accum =
```

```
        let decl = gameobj_decl (chain, name) in
        match decl.B.semant.A.Gameobj.parent with
        | None -> decl :: accum
        | Some (pchain, pname) ->
          helper ((chain @ pchain), pname) (decl :: accum)
      in
      helper (chain, name) []
    in

    let find_obj_event_decl (chain, oname) event =
      try StringMap.find event (gameobj_decl (chain, oname)).B.events
      with Not_found -> failwith ("event " ^ oname ^ "." ^ event)
    in

    let obj_head (chain, oname) =
      try (gameobj_decl (chain, oname)).B.head
      with Not_found -> failwith ("end " ^ oname)
    in

    (* Given value ll for an object of type objname, builds and returns scope
of
       that object in StringMap. *)
    let gameobj_members objref oname builder =
      let add_member llobj (map, ind) (name, typ) =
        let member_var = L.build_struct_gep llobj ind name builder in
        (StringMap.add name (member_var, typ) map, ind + 1)
      in
      let objptr = L.build_extractvalue objref 1 "objptr_gen" builder in

      (* Builds the StringMap assuming this object had name (chain, objname).
         Recurses for inheritance. *)
      let helper parent_map g =
        let llobj =
          L.build_bitcast objptr (L.pointer_type g.B.gtyp) "objptr" builder
        in
        let members = g.B.semant.A.Gameobj.members in
        let (members, _) =
          (* object type struct: gameobj_t :: node_t :: members *)
          List.fold_left (add_member llobj) (parent_map, 2) members
        in
        members
      in
      List.fold_left helper StringMap.empty (inheritance_chain oname)
    in

    (* What we really need is to augment gameobj_methods to also return
       virtual method pointers from the vtable. *)
    let gameobj_methods oname =
```

```
      List.fold_left (fun acc g -> StringMap.fold StringMap.add g.B.methods
acc)
        StringMap.empty (inheritance_chain oname)
    in


    let build_object_loop builder the_function (chain, objname) ~body =
      let g = gameobj_decl (chain, objname) in
      let body builder break_bb node =
        (* Calculating offsets to get the object pointer *)
        let obj = build_container_of g.B.gtyp 1 node objname context builder
~cast:gameobj_t in

        let pred builder _ =
          (* Get values for the removal check *)
          let id = L.build_load (L.build_struct_gep obj 2 "id_ptr" builder)
"id" builder in
          build_print "%d" [id] builder;
          L.build_icmp L.Icmp.Ne id (L.const_null i64_t) "is_removed" builder
        in
        let then_ builder _ = body builder break_bb obj in
        build_if ~pred ~then_ context builder the_function
      in
      build_node_loop builder the_function ~head:(obj_head (chain, objname))
~body
    in

    (* Construct code for an expression used for assignment; return its value
*)
    let rec lexpr (vscope, fscope) builder = function
      | A.Id (chain, n) ->
        (match chain with
         | [] -> let v, _ = StringMap.find n vscope in v
         | _ -> let v, _ = StringMap.find n (namespace_of_chain
chain).B.variables in v)
      | A.Member(e, objname, n) ->
        let vscope = gameobj_members (expr (vscope, fscope) builder e)
objname builder in
        lexpr (vscope, fscope) builder (A.Id([], n))
      | A.Assign (l, r) ->
        let l', r' = lexpr (vscope, fscope) builder l, expr (vscope, fscope)
builder r in
        ignore (L.build_store r' l' builder); l'
      | A.Subscript (arr, ind) ->
        let arr' = lexpr (vscope, fscope) builder arr in
        let ind' = expr (vscope, fscope) builder ind in
        L.build_gep arr' [|L.const_null i32_t; ind'|] "subscript" builder
      | A.Asnop (l, asnop ,t, r) ->
```

```
        let lp = lexpr (vscope, fscope) builder l in
        let le = L.build_load lp "le" builder in
        let re = expr(vscope, fscope) builder r in
        let sum =
          (match t, asnop with
           | A.Int, A.Addasn  -> L.build_add  | A.Float, A.Addasn  ->
L.build_fadd
           | A.Int, A.Subasn  -> L.build_sub  | A.Float, A.Subasn  ->
L.build_fsub
           | A.Int, A.Multasn -> L.build_mul  | A.Float, A.Multasn ->
L.build_fmul
           | A.Int, A.Divasn  -> L.build_sdiv | A.Float, A.Divasn  ->
L.build_fdiv
           | _ -> assert false) le re "Asn" builder
        in
        ignore (L.build_store sum lp builder); lp
      | A.Idop (idop, t, l) ->
        lexpr (vscope, fscope) builder
          (match t, idop with
           | A.Int, A.Inc -> A.Asnop (l, A.Addasn, A.Int, A.Literal 1)
           | A.Int, A.Dec -> A.Asnop (l, A.Subasn, A.Int, A.Literal 1)
           | A.Float, A.Inc -> A.Asnop(l, A.Addasn, A.Float, A.FloatLit 1.0)
           | A.Float, A.Dec -> A.Asnop(l, A.Subasn, A.Float, A.FloatLit 1.0)
           | _ -> assert false)
      | _ -> assert false (* Semant should catch other illegal attempts at
assignment *)
    (* Construct code for an expression; return its value *)
    and expr scope builder = function
      | A.Literal i -> L.const_int i32_t i
      | A.BoolLit b -> L.const_int i1_t (if b then 1 else 0)
      | A.StringLit l -> L.build_global_stringptr l "literal" builder
      | A.FloatLit f -> L.const_float float_t f
      | A.Noexpr -> L.const_int i32_t 0
      | A.NoneObject -> L.const_null objref_t
      | A.Id (_, n) | A.Member (_, _, n) as e -> L.build_load (lexpr scope
builder e) n builder
      | A.Conv (t1, e, t2) ->
        let e' = expr scope builder e in
        (match t1, t2 with
           A.Float, A.Int -> L.build_sitofp e' float_t "" builder
         | A.Int, A.Float -> L.build_fptosi e' i32_t "" builder
         | A.Object _, A.Object _ -> e' (* Do nothing since object lltypes
are all the same *)
         | _ -> assert false)
      | A.Assign _ | A.Asnop _ | A.Idop _ as e -> L.build_load (lexpr scope
builder e) "" builder
      | A.ArrayLit l ->
        let lll = List.map (expr scope builder) l in
```

```
        let typ = (L.array_type (L.type_of (List.hd lll)) (List.length l)) in
        let lll = List.map (fun x -> Some x) lll in
        build_struct_assign (L.undef typ) (Array.of_list lll) builder
      | A.Binop (e1, op, t, e2) ->
        let e1' = expr scope builder e1
        and e2' = expr scope builder e2 in
        (match t with
         | A.Int ->
           (match op with
            | A.Add     -> L.build_add | A.Sub -> L.build_sub
            | A.Mult    -> L.build_mul | A.Div -> L.build_sdiv | A.Modulo ->
L.build_srem
            | A.Equal   -> 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
            | _         -> assert false)
         | A.Float ->
           (match op with
            | A.Add     -> L.build_fadd | A.Sub -> L.build_fsub
            | A.Mult    -> L.build_fmul | A.Div -> L.build_fdiv | A.Modulo ->
L.build_frem
            (* TODO: cut exponent from LRM *)
            | A.Equal   -> L.build_fcmp L.Fcmp.Oeq | A.Neq  -> L.build_fcmp
L.Fcmp.One
            | A.Less    -> L.build_fcmp L.Fcmp.Olt | A.Leq -> L.build_fcmp
L.Fcmp.Ole
            | A.Greater -> L.build_fcmp L.Fcmp.Ogt | A.Geq -> L.build_fcmp
L.Fcmp.Oge
            | _         -> assert false)
         | A.Bool ->
           (match op with (* TODO: SHOULD WE SHORT CIRCUIT? *)
            | A.And -> L.build_and | A.Or -> L.build_or | _ -> assert false)
         | A.Object _ ->
           (match op with
            | A.Equal -> build_obj_cmp ~eq:true | A.Neq -> build_obj_cmp
~eq:false
            | _ -> assert false)
         | _ -> assert false) e1' e2' "tmp" builder
      | A.Unop(op, t, e) ->
        let e' = expr scope builder e in
        (match op with
           A.Neg     -> if t=A.Int then L.build_neg else L.build_fneg
         | A.Not     -> L.build_not) e' "tmp" builder
      | A.Subscript (arr, ind) ->
        let arr', ind' = expr scope builder arr, expr scope builder ind in
```

```
        let arr_ptr = L.build_alloca (L.type_of arr') "arr" builder in
        ignore (L.build_store arr' arr_ptr builder);
        L.build_load (L.build_gep arr_ptr [|L.const_null i32_t; ind'|] ""
builder) "subscript" builder
      | A.Call ((chain, f), act) ->
        let _, fscope = scope in
        let fn =
          match chain with
          | [] -> StringMap.find f fscope
          | _ -> StringMap.find f (namespace_of_chain chain).B.functions
        in
        (* TODO: can arguments have side effects? what's the order-currently
LtR *)
        let actuals = List.map (expr scope builder) act in
        let actuals =             (* Add 'this' argument if member function *)
          match fn.B.gameobj with
          | Some _ -> (expr scope builder (A.Id([], "this"))) :: actuals
          | None -> actuals
        in
        let result =
          match fn.B.return with A.Void -> "" | _ -> f ^ "_result"
        in
        L.build_call fn.B.value (Array.of_list actuals) result builder
      | A.MemberCall (e, objname, f, act) ->
        let fn = StringMap.find f (gameobj_methods objname) in
        let obj = expr scope builder e in
        let actuals = List.map (expr scope builder) act in
        let result =
          match fn.B.return with A.Void -> "" | _ -> f ^ "_result"
        in
        L.build_call fn.B.value (Array.of_list (obj :: actuals)) result
builder
      | A.Create ((chain, objname), args) ->
        let g = gameobj_decl (chain, objname) in
        (* Allocate memory for the object *)
        let llobj = L.build_malloc g.B.gtyp objname builder in
        (* Uncountably many hours have been lost over bugs from uninitialized
           variables... So I'm zero-initing everything. *)
        ignore (L.build_store (L.const_null g.B.gtyp) llobj builder);
        let llobj_gen = L.build_bitcast llobj objptr_t (objname ^ "_gen")
builder in
        (* Set up linked list connections *)
        let make_node_cons (llobj, objname) g =
          let llobjnode = L.build_struct_gep llobj 1 (objname ^ "_objnode")
builder in
          let marker = L.build_struct_gep llobjnode 2 "marker" builder in
          ignore (L.build_store (L.const_int i1_t 1) marker builder);
          let obj_head = g.B.head in
```

```
          ignore (L.build_call list_add_func [|llobjnode; obj_head|] ""
builder);
          let pobjname =
            match g.B.semant.A.Gameobj.parent with
            | None -> "" | Some (_, pobjname) -> pobjname
          in
          L.build_struct_gep llobj 0 (objname ^ "_parent") builder, pobjname
        in
        let chain_from_bottom = List.rev (inheritance_chain (chain, objname))
in
        ignore (List.fold_left make_node_cons (llobj, objname)
chain_from_bottom);
        (* As long as generic gameobj is parent of all, we don't need to also
           specifically set its connections. *)
        (* let llnode = L.build_struct_gep llobj_gen 1 (objname ^ "_node")
builder in *)
        (* ignore (L.build_call list_add_func [|llnode; gameobj_head|] ""
builder); *)
        (* Update ID and ID counter *)
        let llid = L.build_load global_objid "old_id" builder in
        let llid = L.build_add llid (L.const_int i64_t 1) "new_id" builder in
        ignore (L.build_store llid global_objid builder);
        (* Event function pointers *)
        build_struct_ptr_assign llobj_gen [|Some g.B.vtable; None; Some llid|]
builder;
        let objref = build_struct_assign (L.undef objref_t) [|Some llid; Some
llobj_gen|] builder in
        let create_event = (find_obj_event_decl (chain, objname)
"create").B.value in
        (* TODO: include something in LRM about non-initialized values *)
        let actuals = List.map (expr scope builder) args in
        ignore (L.build_call create_event (Array.of_list (objref :: actuals))
"" builder);
        objref
      | A.Destroy e ->
        let objref = expr scope builder e in
        ignore (build_delete objref builder ~destroy:true);
        expr scope builder (A.Noexpr) (* considered Void in semant *)
      | A.Delete e ->
        let objref = expr scope builder e in
        ignore (build_delete objref builder ~destroy:false);
        expr scope builder (A.Noexpr) (* considered Void in semant *)
    in

    (* Build the code for the given statement, given the following:
       - the encapsulating function llvalue
       - the block to jump to in the case of a break
       - the return type
```

```
       - the builder to start at
       - the current scope.
       Returns the builder for the statement's successor. Builder is
       guaranteed to point to a block without a terminator. *)
    let rec stmt fn break_bb ret_t (builder, scope) = function
      | A.Decl (name, typ) ->
        let vscope, fscope = scope in
        let local_var = L.build_alloca (ltype_of_typ typ) name builder in
        builder, (StringMap.add name (local_var, typ) vscope, fscope)
      | A.Vdef ((name, typ), e) ->
        let vscope, fscope = scope in
        let local_var = L.build_alloca (ltype_of_typ typ) name builder in
        let new_vscope = StringMap.add name (local_var, typ) vscope in
        let e' = expr (vscope, fscope) builder e in
        ignore(L.build_store e' local_var builder);
        builder, (StringMap.add name (local_var, typ) new_vscope, fscope)
      | A.Block b ->
        let merge_bb = L.append_block context "block_end" fn in
        let builder, _ =
          List.fold_left (stmt fn break_bb ret_t) (builder, scope) b
        in
        ignore (L.build_br merge_bb builder);
        L.builder_at_end context merge_bb, scope
      | A.Expr e -> ignore (expr scope builder e); builder, scope
      | A.Break -> ignore (L.build_br break_bb builder);
        let dead_bb = L.append_block context "postbreak" fn in
        L.builder_at_end context dead_bb, scope
      | A.Return e ->
        ignore (match ret_t with
            | A.Void -> L.build_ret_void builder
            | _ -> L.build_ret (expr scope builder e) builder);
        let dead_bb = L.append_block context "postret" fn in
        L.builder_at_end context dead_bb, scope
      | A.If (predicate, then_stmt, else_stmt) ->
        let run st b _ = fst (stmt fn break_bb ret_t (b, scope) st) in
        build_if context builder fn ~then_:(run then_stmt) ~else_:(run
else_stmt)
          ~pred:(fun b _ -> expr scope b predicate),
        scope
      | A.While (predicate, body) ->
        build_while context builder fn
          ~pred:(fun b _ -> expr scope b predicate, ())
          ~body:(fun b br _ () -> fst (stmt fn br ret_t (b, scope) body)),
scope
      | A.For (s1, e2, e3, body) ->
        let while_stmts = [s1; A.While (e2, A.Block [body; A.Expr e3])] in
        let for_builder, _ =
          stmt fn break_bb ret_t (builder, scope) (A.Block while_stmts)
```

```
        in
        for_builder, scope
      | A.Foreach ((name, typ), body_stmt) ->
        match typ with
        | A.Object objname ->
          (* TODO: describe semantics. what if obj of type is destroyed or
created in this? *)
          let body builder break_bb objptr =
            let id = L.build_load (L.build_struct_gep objptr 2 "" builder)
"id" builder in
            let obj = build_struct_assign (L.undef objref_t) [|Some id; Some
objptr|] builder in
            let objref = L.build_alloca objref_t "ref" builder in
            ignore (L.build_store obj objref builder);
            let builder, _ =
              let vscope, fscope = scope in
              let vscope' = StringMap.add name (objref, A.Object(objname))
vscope in
              stmt fn break_bb ret_t (builder, (vscope', fscope)) body_stmt
            in
            builder
          in
          build_object_loop builder fn objname ~body, scope
        | _ -> assert false
    in

    let build_function_body the_function formals block ?gconfig return_type =
      let entry = L.entry_block the_function in
      let builder = L.builder_at_end context entry in

      (* If this is a function in a gameobj, add 'this' to the arguments *)
      let formals =
        match gconfig with
        | Some { Gconfig.obj; _ } -> ("this", A.Object([], obj)) :: formals
        | None -> formals
      in
      let formal_scope =
        let add_formal m (n, t) 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, t) m
        in
        List.fold_left2 add_formal StringMap.empty formals
          (Array.to_list (L.params the_function))
      in
      let member_scope =
        match gconfig with
```

```
        | Some { Gconfig.obj; _ } ->
          let this, _ = StringMap.find "this" formal_scope in
          let this = L.build_load this "thisref" builder in
          gameobj_members this ([], obj) builder
        | None -> StringMap.empty
      in
      let method_scope =
        match gconfig with
        | None -> StringMap.empty
        | Some { Gconfig.obj; super; _ } ->
          let g = StringMap.find obj llns.B.gameobjs in
          match g.B.semant.A.Gameobj.parent, super with
          | None, _ | _, None -> gameobj_methods ([], obj)
          | Some (chain, pname), Some super_method ->
            (* Add super(...) as a way to call the parent method, if allowed.
*)
            let pobj = gameobj_decl (chain, pname) in
            gameobj_methods ([], obj)
            |> StringMap.add "super" (StringMap.find super_method
pobj.B.events)
      in
      let add_to_scope to_add = StringMap.fold StringMap.add to_add in
      let scope =
        llns.B.variables |> add_to_scope member_scope |> add_to_scope
formal_scope,
        llns.B.functions |> add_to_scope method_scope
      in

      let builder, _ =
        stmt the_function entry return_type (builder, scope) (A.Block(block))
      in

      (* Possibly add some code at the end *)
      let builder = match gconfig with
        | None -> builder
        | Some { Gconfig.postwork; _ } ->
          match postwork with Some p -> p builder scope | None -> builder
      in

      (* Add a precautionary return to the end *)
      ignore (match return_type with
          | A.Void -> L.build_ret_void builder
          | t -> L.build_ret (L.const_null (ltype_of_typ t)) builder)
    in

    let build_function (fname, { A.Func.block; formals; typ; gameobj = _ }) =
      match block with
      | Some block ->
```

```
        let llfn = (StringMap.find fname (llns.B.functions)).B.value in
        build_function_body llfn formals block typ
      | None -> ()
    in
    List.iter build_function functions;

    let build_obj_functions (gname, g) =
      let open A.Gameobj in

      let build_fn find_fn (fname, { A.Func.typ; formals; block; gameobj =
_ }) =
        let llfn = (find_fn ([], gname) fname).B.value in
        match block with
        | Some block ->
          (* Add super into scope on case-by-case basis. Casework used to be
more complicated. *)
          let super = match fname with "create" | "step" | "draw" | "destroy"
-> Some fname | _ -> None in
          let gconfig = { Gconfig.obj = gname; postwork = None; super } in
          build_function_body llfn formals block typ ~gconfig
        | None -> assert false    (* Semant ensures obj fns are not external
*)
      in
      let find_obj_fn_decl (chain, oname) fname =
        StringMap.find fname (gameobj_decl (chain, oname)).B.methods
      in
      List.iter (build_fn find_obj_fn_decl) g.methods;
      List.iter (build_fn find_obj_event_decl) g.events;
    in
    List.iter build_obj_functions gameobjs;

    (* Build global_create from the global namespace *)
    if nname = "" then
      let create_gb = L.define_function "global_create" (L.function_type
void_t [||]) the_module in
      build_function_body create_gb [] [A.Expr (A.Create (([], "main"), []))]
A.Void
    else ()
  in

  let global_event (name, offset) =
    let fn = L.define_function ("global_" ^ name) (L.function_type void_t
[||]) the_module in
    let builder = L.builder_at_end context (L.entry_block fn) in
    let body builder _ node =
      let objptr = build_container_of gameobj_t 1 node "objptr" context
builder in
```

```
      let id = L.build_load (L.build_struct_gep objptr 2 "id_ptr" builder)
"id" builder in

      let pred b _ = L.build_icmp L.Icmp.Ne id (L.const_null i64_t)
"is_removed" b in

      let call b _ =
        let this_fn =
          let tbl = L.build_load (L.build_struct_gep objptr 0 "" b)
("this_tbl") b in
          let ptr = L.build_struct_gep tbl offset ("this_" ^ name ^ "ptr") b
in
          L.build_load ptr ("this_" ^ name) b
        in
        let objref = build_struct_assign (L.undef objref_t) [|Some id; Some
objptr|] b in
        ignore (L.build_call this_fn [|objref|] "" b); b
      in

      let remove b _ =
        ignore (L.build_call list_rem_func [|node|] "" b);
        ignore (L.build_store (L.const_null gameobj_t) objptr b); (* for
safety/faster errors *)
        ignore (L.build_free objptr b); b
      in

      build_if ~pred ~then_:call ~else_:remove context builder fn
    in
    let builder = build_node_loop builder fn ~head:gameobj_head ~body in
    ignore (L.build_ret_void builder)
  in
  List.iter global_event ["step", 0; "draw", 2];

  define_ns_contents llprogram ("", the_program);
  List.iter (fun (fname, ns) -> define_ns_contents (StringMap.find fname
llfiles) (fname, ns))
    files;

  the_module
```

makergame.ml

```
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
  (* print_string (Ast.string_of_program ast); *)
  match action with
    Ast -> print_string (Ast.string_of_program ast)
  | LLVM_IR ->
    let { Ast.main = sast; Ast.files = files } = Semant.check_program ast in
    print_string (Llvm.string_of_llmodule (Codegen.translate sast files))
  | Compile ->
    let { Ast.main = sast; Ast.files = files } = Semant.check_program ast in
    let m = Codegen.translate sast files in
    Llvm_analysis.assert_valid_module m;
    print_string (Llvm.string_of_llmodule m)
```

External Runtime Library (Steven, Cindy)

libmakergame.cpp (Steven)

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <map>
#include <string>
#include <iostream>
#include <cstdlib>

static std::map<std::string, sf::Texture> image_map;
static std::map<std::string, sf::Sprite> sprite_map;
static std::map<std::string, sf::SoundBuffer> sound_map;
static sf::RenderWindow window;
static sf::Color clear_color = sf::Color(255, 255, 255);

static void display_window(sf::RenderWindow *window) { window->display(); }

static sf::RenderWindow *create_window(int width, int height,
                                       const char *wname) {
  return new sf::RenderWindow(sf::VideoMode(width, height), wname);
}

static void close_window(sf::RenderWindow *window) {
  if (window->isOpen()) window->close();
}

static void play_sound(sf::Sound *sound, bool loop) {
  sound->setLoop(loop);
  sound->play();
```

```
}

static bool game_ended = false;

extern "C" {

// built-in print functions
void print(int x) { printf("%d\n", x); }
void printb(bool x) { if (x) printf("true\n"); else printf("false\n"); }
void print_float(double x) { printf("%f\n", x); }
void printstr(char *x) { printf("%s\n", x); }

sf::Sound *load_sound(const char *filename) {
  if (!sound_map.count(filename) && !
sound_map[filename].loadFromFile(filename))
    std::cerr << "unable to load sound " << filename << "\n";
  return new sf::Sound(sound_map[filename]);
}

void set_window_size(int w, int h) {
  window.setSize(sf::Vector2u(w, h));
  window.setView(sf::View(sf::FloatRect(0, 0, w, h)));
}
void set_window_clear(int r, int g, int b) { clear_color = sf::Color(r, g,
b); }
void set_window_title(char *x) { window.setTitle(x); }

void play_sound(sf::Sound *sound) { play_sound(sound, false); }
void loop_sound(sf::Sound *sound) { play_sound(sound, true); }
void stop_sound(sf::Sound *sound) { sound->stop(); }

sf::Sprite *load_image(const char *filename) {
  if (!image_map.count(filename)) {
    if (!image_map[filename].loadFromFile(filename))
      std::cerr << "unable to load image " << filename << "\n";
    sprite_map[filename] = sf::Sprite(image_map[filename]);
  }
  return &sprite_map[filename];
}

int sprite_width(sf::Sprite *sprite) { return sprite->getTextureRect().width;
}
int sprite_height(sf::Sprite *sprite) { return sprite-
>getTextureRect().height; }

void draw_sprite_rect(sf::Sprite *sprite, double x, double y, int sx, int sy,
int sw, int sh) {
  sprite->setPosition(x, y);
```

```
  sprite->setTextureRect(sf::IntRect(sx, sy, sw, sh));
  window.draw(*sprite);
}

void draw_sprite(sf::Sprite *sprite, double x, double y) {
  sprite->setPosition(x, y);
  sprite->setTexture(*sprite->getTexture(), true); // reset texture rect
  window.draw(*sprite);
}

void end_game() { close_window(&window); game_ended = true; }

int rand_max() { return RAND_MAX; }

bool key_pressed(int code) {
  return sf::Keyboard::isKeyPressed(sf::Keyboard::Key(code));
}
// TODO: add mouse function. should be pretty easy with sf::Mouse

void global_create();
void global_step();
void global_draw();
}

int main() {
  srand(time(NULL));

  window.create(sf::VideoMode(800, 600), "MakerGame Game");
  window.setFramerateLimit(60);

  global_create();

  while (window.isOpen()) {
    sf::Event event;
    while (window.pollEvent(event)) {
      if (event.type == sf::Event::Closed)
        window.close();
    }
    global_step();
    window.clear(clear_color);
    global_draw();
    window.display();
  }

  sound_map.empty();
  image_map.empty();

  return 0;
```

```
}
```

libtestergame.cpp (Steven, Cindy)

```
// SFML-independent environment for testing
#include <cstdio>
static bool game_ended = false;
static const int max_steps = 100;

extern "C" {

// built-in print functions
void print(int x) { printf("%d\n", x); }
void printb(bool x) { if (x) printf("true\n"); else printf("false\n"); }
void print_float(double x) { printf("%f\n", x); }
void printstr(char *x) { printf("%s\n", x); }

// dummy remainder functions
void *load_sound(const char *filename) { return nullptr; }
void play_sound(void *sound) { }
void loop_sound(void *sound) { }
void stop_sound(void *sound) { }
void *load_image(const char *filename) { return nullptr; }
void set_sprite_position(void *s, double x, double y) { }
void draw_sprite(void *sprite) { }
void draw_sprite_rect(void *sprite, double x, double y, int sx, int sy, int
sw, int sh) { }
void end_game() { game_ended = true; }
bool key_pressed(int code) { return false; }

void global_create();
void global_step();
void global_draw();

int rand_max() { return 0; }

void set_window_size(int w, int h) { }
void set_window_clear(int r, int g, int b) { }
void set_window_title(char *x) { }

int sprite_width() { return 0; }
int sprite_height() { return 0; }

}

int main() {
  global_create();
```

```
  int num_steps = 0;
  while (!game_ended) {
    global_step();
    global_draw();
    ++num_steps;
    if (num_steps >= max_steps) {
      fprintf(stderr, "FAILURE: Exceed max number of steps allowed for test.
"
              "Did you forget to call end_game()?\n");
      return 1;
    }
  }

  return 0;
}
```

MakerGame Standard Library (Steven)

std.mg

```
namespace print  = open "print.mg";
namespace spr    = open "sprite.mg";
namespace snd    = open "sound.mg";
namespace math   = open "math.mg";
namespace window = open "window.mg";
namespace key    = open "key.mg";
namespace game   = open "game.mg";
```

math.mg

```
private namespace p {
  extern int rand();
  extern int rand_max();
}

int irandom(int x) { return p::rand() % x; }
float frandom() {
  float x = p::rand();
  float y = p::rand_max();
  return x / y;
}

// Trig, straight from cmath, linked via -lm
extern float sin(float x);
extern float cos(float x);
extern float tan(float x);
extern float asin(float x);
extern float acos(float x);
```

```
extern float atan(float x);
extern float atan2(float y, float x);
extern float sqrt(float x);
```

print.mg

```
private namespace p {
  extern void print(int x);
  extern void printb(bool x);
  extern void print_float(float x);
  extern void printstr(string x);
}

void i(int x) { p::print(x); }
void b(bool x) { p::printb(x); }
void f(float x) { p::print_float(x); }
void s(string x) { p::printstr(x); }
```

sound.mg

```
private namespace p {
  extern sound load_sound(string filename);
  extern void play_sound(sound snd, bool loop);
  extern void stop_sound(sound snd);
}

sound load(string filename) { return p::load_sound(filename); }
void play(sound s) { p::play_sound(s, false); }
void loop(sound s) { p::play_sound(s, true); }
void stop(sound s) { p::stop_sound(s); }
```

sprite.mg

```
private namespace p {
  extern sound load_sound(string filename);
  extern void play_sound(sound snd, bool loop);
  extern void stop_sound(sound snd);
}

sound load(string filename) { return p::load_sound(filename); }
void play(sound s) { p::play_sound(s, false); }
void loop(sound s) { p::play_sound(s, true); }
void stop(sound s) { p::stop_sound(s); }
```

key.mg

```
int A         = 0;
int B         = 1;
int C         = 2;
```

```
int D         = 3;
int E         = 4;
int F         = 5;
int G         = 6;
int H         = 7;
int I         = 8;
int J         = 9;
int K         = 10;
int L         = 11;
int M         = 12;
int N         = 13;
int O         = 14;
int P         = 15;
int Q         = 16;
int R         = 17;
int S         = 18;
int T         = 19;
int U         = 20;
int V         = 21;
int W         = 22;
int X         = 23;
int Y         = 24;
int Z         = 25;
int Num0      = 26;
int Num1      = 27;
int Num2      = 28;
int Num3      = 29;
int Num4      = 30;
int Num5      = 31;
int Num6      = 32;
int Num7      = 33;
int Num8      = 34;
int Num9      = 35;
int Escape    = 36;
int LControl  = 37;
int LShift    = 38;
int LAlt      = 39;
int LSystem   = 40;
int RControl  = 41;
int RShift    = 42;
int RAlt      = 43;
int RSystem   = 44;
int Menu      = 45;
int LBracket  = 46;
int RBracket  = 47;
int SemiColon = 48;
int Comma     = 49;
int Period    = 50;
```

```
int Quote     = 51;
int Slash     = 52;
int BackSlash = 53;
int Tilde     = 54;
int Equal     = 55;
int Dash      = 56;
int Space     = 57;
int Return    = 58;
int BackSpace = 59;
int Tab       = 60;
int PageUp    = 61;
int PageDown  = 62;
int End       = 63;
int Home      = 64;
int Insert    = 65;
int Delete    = 66;
int Add       = 67;
int Subtract  = 68;
int Multiply  = 69;
int Divide    = 70;
int Left      = 71;
int Right     = 72;
int Up        = 73;
int Down      = 74;
int Numpad0   = 75;
int Numpad1   = 76;
int Numpad2   = 77;
int Numpad3   = 78;
int Numpad4   = 79;
int Numpad5   = 80;
int Numpad6   = 81;
int Numpad7   = 82;
int Numpad8   = 83;
int Numpad9   = 84;
int F1        = 85;
int F2        = 86;
int F3        = 87;
int F4        = 88;
int F5        = 89;
int F6        = 90;
int F7        = 91;
int F8        = 92;
int F9        = 93;
int F10       = 94;
int F11       = 95;
int F12       = 96;
int F13       = 97;
int F14       = 98;
```

```
int F15       = 99;
int Pause     = 100;

private namespace p {
  extern bool key_pressed(int x);

  // Define an object to store the state of key presses last step.
  // This way we can detect when a key is just pressed or just released.
  // TODO: have some way to get this into the game by default!
  object Checker {
    bool pressed[101];
    event create {
      for (int i = 0; i < 101; ++i) pressed[i] = false;
    }
    event draw {
      for (int i = 0; i < 101; ++i) pressed[i] = key_pressed(i);
    }
  }
  Checker c;
}

bool is_down(int x) { return p::key_pressed(x); }
bool is_pressed(int x) {
  return (p::key_pressed(x) && !p::c.pressed[x]); // down this frame but not
last
}

bool is_released(int x) {
  return (!p::key_pressed(x) && p::c.pressed[x]); // down last frame but not
this
}

void set_key() {
  // check if checker exists. if not create
  p::c = create p::Checker;
}
```

window.mg

```
private namespace p {
  extern void set_window_size(int w, int h);
  extern void set_window_clear(int r, int g, int b);
  extern void set_window_title(string x);
}

void set_size(int w, int h) { p::set_window_size(w, h); }
void set_clear(int r, int g, int b) { p::set_window_clear(r, g, b); }
void set_title(string title) { p::set_window_title(title); }
```

game.mg

```
namespace key = open "key.mg";
namespace spr = open "sprite.mg";
namespace snd = open "sound.mg";

private namespace p {
  extern void end_game();
}

bool obj_alive (object o) {
  foreach (object x)
    if (x == o) return true;
  return false;
}

// objects that remove all other objects
// TODO: Should remove at start of step. but awkward syntax then...
//       To get this right, need real custom virtual functions
object room {
  event create {
    foreach (object r) { if (r != this) delete r; }
    key::set_key();
  }
}

object obj {
  sprite spr;
  // TODO: syntactic sugar for vec2i, vec2f. maybe vec3 too
  // basically let x and y be [0] and [1] respectively for float[2]
  // rects could be float[4], w and h the next [2] and [3]
  float x; float y;
  float hspeed; float vspeed;
  float origin_x; float origin_y;
  float hitbox_offx; float hitbox_offy; float hitbox_w; float hitbox_h;

  // center the hitbox on the origin with width sw, height sh
  void center_hitbox_abs(float sw, float sh) {
    set_hitbox(-sw/2, -sh/2, sw, sh);
  }

  // centre the hitbox on the origin, and set its width and height to
  // xprop * sprite_width, yprop * sprite_height
  void center_hitbox_prop(float xprop, float yprop) {
    float sw = xprop * spr::width(spr);
    float sh = yprop * spr::height(spr);
    center_hitbox_abs(sw, sh);
  }
```

```
  void set_hitbox(float x, float y, float w, float h) {
    hitbox_offx = x; hitbox_offy = y;
    hitbox_w = w; hitbox_h = h;
  }

  event create(float x, float y, string sprite_name) {
    this.x = x; this.y = y;
    spr = spr::load(sprite_name);
    origin_x = spr::width(spr) * 0.5;
    origin_y = spr::height(spr) * 0.5;
    center_hitbox_abs(0, 0);
  }
  event draw {
    x += hspeed; y += vspeed;
    spr::render(spr, x-origin_x, y-origin_y);
  }
}

bool colliding(obj a, obj b) {
  float al = a.x + a.hitbox_offx;
  float au = a.y + a.hitbox_offy;
  float ar = a.x + a.hitbox_offx + a.hitbox_w;
  float ad = a.y + a.hitbox_offy + a.hitbox_h;
  float bl = b.x + b.hitbox_offx;
  float bu = b.y + b.hitbox_offy;
  float br = b.x + b.hitbox_offx + b.hitbox_w;
  float bd = b.y + b.hitbox_offy + b.hitbox_h;
  return (al < br && ar > bl && au < bd && ad > bu);
}

void end() { p::end_game(); }
```

Egg Demo (Steven)

egg.mg

```
// make relevant namespaces more easily accessible
// the best way is slightly verbose right now, since
// "using" does not make inner namespaces available
// (for deeper language design reasons... since declaration
//  order does not matter, it will be tough to decide how
//  to get using inner inner namespaces after using an inner
//  namespace.)
namespace spr    = std::spr;
namespace snd    = std::snd;
namespace key    = std::key;
```

```
namespace window = std::window;
namespace game   = std::game;

namespace Num = open "draw_numbers.mg";
Num::Draw numbers;

sound boinkSound;
int score;

object Egg : game::obj {
  int points;

  event create(float x, float y) {
    super(x, y, "egg.png");
    vspeed = 3 + std::math::frandom() * (1.4 + times * 0.02);
    points = vspeed * 10;
    snd::play(boinkSound);
    center_hitbox_prop(0.9, 0.9);
  }

  event step {
    if (y > 600) create gameover; // go to the room
  }
}

object SineEgg : Egg {
  int timer;

  event create(float x, float y) {
    super(x, y);
    points *= 1.5;
    timer = 0;
    spr = spr::load("flying-egg.png");
    hitbox_offx = -spr::width(spr)/2;
    hitbox_offy = -spr::height(spr)/2;
  }

  event step {
    super();
    ++timer;
    hspeed = 5 * std::math::sin(timer * 0.1);
  }
}

object Player : game::obj {
  event create {
    super(300, 500, "player.png");
    center_hitbox_prop(0.9, 0.6);
```

```
  }

  event step {
    if (key::is_down(key::Left)) x -= 5;
    if (key::is_down(key::Right)) x += 5;

    foreach (Egg egg) {
      if (game::colliding(egg, this)) {
        score += egg.points;
        numbers.n = score;
        destroy egg;
      }
    }
  }
}

int times = 0; // to keep track of progress
object Spawner {
  int timer;
  event create { timer = 50; times = 0; }
  event step {
    --timer;
    if (timer == 0) {
      timer = 50 - times/8;
      if (timer < 8) timer = 8;
      ++times;
      int range = 300;
      int x = 400 + (range * std::math::frandom() - range * 0.5);
      if (std::math::irandom(5) == 0) create SineEgg(x, 0);
      else create Egg(x, 0);
    }
  }
}

object main : game::room {
  event create {
    super();
    window::set_title("egg game");
    score = 0;
    boinkSound = snd::load("boink.ogx");
    create Player;
    create Spawner;
    numbers = create Num::Draw(0, 10, 10);
  }
}

object gameover : game::room {
  sprite spr;
```

```
  event create {
    super();
    spr = spr::load("gameover.png");
  }
  event draw { spr::render(spr, 0, 0); }
}
```

draw_numbers.mg

```
// makeshift number drawing
using std::spr;

object Draw {
  sprite spr[10];
  float x; float y;
  int n;
  int width;

  event create(int number, float x, float y) {
    this.x = x; this.y = y;
    n = number;
    spr = [load("numbers/0.png"), load("numbers/1.png"),
           load("numbers/2.png"), load("numbers/3.png"),
           load("numbers/4.png"), load("numbers/5.png"),
           load("numbers/6.png"), load("numbers/7.png"),
           load("numbers/8.png"), load("numbers/9.png")];
  }

  int draw_indiv(int n, float X) {
    if (n > 9) X = draw_indiv(n/10, X);
    render(spr[n%10], X, y);
    return (X + width(spr[n%10]));
  }

  event draw {
    int n = this.n;
    if (n < 0) n = 0;
    draw_indiv(n, x);
  }
}
```

egg.ll (generated)

Okay, the generated output is actually extremely verbose since every time we access a member variable, code is generated to access all the member variables (so that codegen can pass around StringMaps of guaranteed llvalues). Presumably all these redundant accesses get optimized out, but for the purposes of the code listing, one of these files adds 70 pages, so it's omitted. What we probably could've done better is instead pass around StringMaps of thunks returning llvalues. In any case, the ll files for the demos are included in the demo directory (makergame/demo/egg/egg.ll and makergame/demo/tetris/tetris.ll)

Tetris Demo (Steven)

tetris.mg

```
namespace spr    = std::spr;
namespace snd    = std::snd;
namespace key    = std::key;
namespace window = std::window;
namespace game   = std::game;

namespace Numbers = open "draw_numbers.mg";
Numbers::Draw piece_counts[7];
Numbers::Draw level_count;
Numbers::Draw top_count;
Numbers::Draw score_count;
Numbers::Draw lines_count;

namespace snds {
  sound clear;
  sound drop;
  sound level;
  sound move;
  sound tetris;
  sound turn;
  sound music;

  namespace snd = std::snd;
  void load() {
    clear  = snd::load("snd/clear.ogg");
    drop   = snd::load("snd/drop.ogg");
    level  = snd::load("snd/level.ogg");
    music  = snd::load("snd/music.ogg");
    move   = snd::load("snd/move.ogg");
    tetris = snd::load("snd/tetris.ogg");
    turn   = snd::load("snd/turn.ogg");
  }
}
```

```
// TODO: make sure path is set
bool possible_pieces[7][4][3] = [
  [[false, true , false], // t
   [true , true , true ],
   [false, false, false],
   [false, false, false]],

  [[true , false, false], // periscope
   [true , true , true ],
   [false, false, false],
   [false, false, false]],

  [[true , true , false], // z
   [false, true , true ],
   [false, false, false],
   [false, false, false]],

  [[true , true , false], // box
   [true , true , false],
   [false, false, false],
   [false, false, false]],

  [[false, true , true ], // s
   [true , true , false],
   [false, false, false],
   [false, false, false]],

  [[false, false, true ], // L
   [true , true , true ],
   [false, false, false],
   [false, false, false]],

  [[false, true , false], // long
   [false, true , false],
   [false, true , false],
   [false, true , false]]];

int level_speed[19] = [48, 43, 38, 33, 28, 23, 18, 13, 8, 6, 5, 5, 5, 4, 4, 4,
3, 3, 3];
int current_speed() {
  int lv = level_count.n;
  if (lv < 19) return level_speed[lv];
  else if (lv < 29) return 2;
  else return 1;
}

int boardOffsetX = 96;
```

```
int boardOffsetY = 40;
int tile_size = 8;

object Block {
  int x; int y;
  int type_x; int type_y;
  int type_rect[4];
  sprite s;
  Piece piece; // need to set piece to none at some point
  event create(int x, int y, Piece p) {
    s = spr::load("img/pieces.png");
    this.x = x; this.y = y; piece = p;
    setType(0, 0);
  }

  void setType(int x, int y) {
    type_x = x; type_y = y;
    type_rect = [tile_size*x, tile_size*y, tile_size, tile_size];
  }

  void rotateLeft() {
    int newX = -y; int newY = x;
    y = newY; x = newX;
  }

  void rotateRight() {
    rotateLeft(); rotateLeft(); rotateLeft();
  }

  void settlePosition(Board board) {
    x = piece.x + x;
    y = piece.y + y;
    piece = none;
    if (y <= 0) { create game_over(score_count.n); return; }
    board.pieces[y][x] = this;
  }

  event draw {
    float draw_x = x; float draw_y = y;
    if (piece != none) {
      if (!piece.active) {
        draw_x = x + 13.5;
        draw_y = y + 9;
      }
      // with an active piece, x and y are relative positions to the piece.
      else {
        draw_x = x + piece.x;
        draw_y = y + piece.y;
```

```
      }
    }
    draw_x *= tile_size; draw_y *= tile_size;
    draw_x += boardOffsetX;
    draw_y += boardOffsetY;
    spr::render_rect(s, draw_x, draw_y, type_rect);
  }
}

object Piece {
  Block blocks[4];
  Board board;
  int x; int y;
  int curr_timer;
  int horiz_timer;
  int drop_points;
  bool active;
  int piece_type;
  int hard_drop;
  Piece next;
  event create(Board b) {
    drop_points = 0;
    board = b;
    curr_timer = current_speed();
    active = false;
    hard_drop = 1;

    int piece_type_x = std::math::irandom(3);
    int piece_type_y = std::math::irandom(10);
    piece_type = std::math::irandom(7);
    int k = 0;
    for (int y = 0; y < 4; ++y)
      for (int x = 0; x < 3; ++x)
        if (possible_pieces[piece_type][y][x]) {
          // centres of pieces are at x = 1, y = 1
          blocks[k] = create Block(x-1, y-1, this);
          blocks[k].setType(piece_type_x, piece_type_y);
          ++k;
        }
  }

  void activate() {
    ++piece_counts[piece_type].n;
    active = true;
    next = create Piece(board);
    x = 5; y = 1;
  }
```

```
  bool isColliding() {
    for (int i = 0; i < 4; ++i)
      if (board.occupied(blocks[i].x + x, blocks[i].y + y))
        return true;
    return false;
  }

  void rotateLeft() {
    for (int i = 0; i < 4; ++i) blocks[i].rotateLeft();
    if (isColliding())
      for (int i = 0; i < 4; ++i) blocks[i].rotateRight();
    else snd::play(snds::turn);
  }

  void rotateRight() {
    for (int i = 0; i < 4; ++i) blocks[i].rotateRight();
    if (isColliding())
      for (int i = 0; i < 4; ++i) blocks[i].rotateLeft();
    else snd::play(snds::turn);
  }

  void moveLeft() { --x; if (isColliding()) ++x; else
snd::play(snds::move); }
  void moveRight() { ++x; if (isColliding()) --x; else snd::play(snds::move);
}
  void moveDown() {
    curr_timer = current_speed();
    ++y;
    if (isColliding()) {
      --y;
      settlePosition();
    }
  }

  void settlePosition() {
    for (int i = 0; i < 4; ++i) {
      blocks[i].settlePosition(board);
      blocks[i].piece = none;
    }
    snd::play(snds::drop);
    board.checkRows();
    active = false;
    destroy this;
    score_count.n += drop_points;
    foreach (game_over g) { return; } // janky: dont create if in gameover
    next.activate();
    snd::play(snds::drop);
  }
```

```
  event step {
    if (!active) return;
    if (key::is_pressed(key::Left) || (key::is_down(key::Left) && horiz_timer
== 0)) {
      horiz_timer = 5;
      moveLeft();
    }
    if (key::is_pressed(key::Right) || (key::is_down(key::Right) &&
horiz_timer == 0)) {
      horiz_timer = 5;
      moveRight();
    }
    if (key::is_down(key::Down)) {
      if (curr_timer > 3) {
        curr_timer = 3; ++drop_points;
      }
    }
    if (key::is_pressed(key::X)) rotateLeft();
    if (key::is_pressed(key::Z)) rotateRight();
    if (key::is_pressed(key::Space) && hard_drop == 0)
      while (active) { ++drop_points; moveDown(); }
    if (--curr_timer == 0) moveDown();
    if (horiz_timer > 0) --horiz_timer;
    if (hard_drop > 0) --hard_drop;
  }
}

int board_width = 10;
int board_height = 20;
int points[5] = [0, 40, 100, 300, 1200];
object Board {
  Block pieces[24][10];

  event create {
    for (int y = 0; y < board_height; ++y)
      for (int x = 0; x < board_width; ++x)
        pieces[y][x] = none;
  }

  void checkRows() {
    int lines = 0;
    for (int y = 0; y < board_height; ++y)
      if (checkRow(y)) ++lines;
    if (lines >= 4)
      snd::play(snds::tetris);
    else if (lines >= 1)
      snd::play(snds::clear);
```

```
    score_count.n += points[lines] * (level_count.n + 1);
  }

  bool checkRow(int r) {
    // check if row full
    for (int x = 0; x < board_width; ++x)
      if (pieces[r][x] == none)
        return false;

    // remove things in row
    for (int x = 0; x < board_width; ++x)
      destroy pieces[r][x];

    // move upper rows down
    for (int y = r; y > 0; --y)
      for (int x = 0; x < board_width; ++x) {
        pieces[y][x] = pieces[y-1][x];
        if (pieces[y][x] != none)
          ++pieces[y][x].y;
      }

    for (int x = 0; x < board_width; ++x)
      pieces[0][x] = none;

    ++lines_count.n;
    if (lines_count.n / 5 > level_count.n) {
      level_count.n = lines_count.n / 5;
      snd::play(snds::level);
    }
    if (level_count.n > 99) level_count.n = 99;
    return true;
  }

  bool occupied(int x, int y) {
    if (x < 0 || x >= board_width || y >= board_height) return true;
    if (y < 0) return false;
    if (pieces[y][x] != none) return true;
    return false;
  }
}

object main : game::room {
  event create {
    super();
    snds::load();
    std::window::set_clear(0, 0, 0);
    std::window::set_title("Tetris");
    std::window::set_size(256, 224);
```

```
  }

  event step {
    if (key::is_pressed(key::Space))
      create game_room;
  }

  event draw {
    spr::render(spr::load("img/title.png"), 0, 0);
  }
}

object game_room : game::room {
  event create {
    super();
    snd::loop(snds::music);
    Board b = create Board;
    for (int i = 0; i < 7; ++i)
      piece_counts[i] = create Numbers::Draw(48, 88+16*i, 3);

    level_count = create Numbers::Draw(208, 160, 2);
    top_count = create Numbers::Draw(192, 32, 6);
    score_count = create Numbers::Draw(192, 56, 6);
    lines_count = create Numbers::Draw(152, 16, 3);

    (create Piece(b)).activate();
  }

  event step {
    top_count.n = score_count.n;
    super();
  }

  event draw {
    std::spr::render(spr::load("img/board.png"), 0, 0);
  }
}

int game_over_fade = 120;
int game_over_delay = 120;
object game_over : game::room {
  int timer;
  int n;
  event create(int score) {
    super();
    n = score;
    window::set_clear(0, 0, 0);
    snd::stop(snds::music);
```

```
  }

  event step {
    ++timer;
    if (timer == game_over_fade)
      score_count = create Numbers::Draw(104, 136, 6);
    if (timer >= game_over_fade + game_over_delay && score_count.n < n) {
      int add = n/game_over_delay/2 + 1;
      score_count.n += add;
      if (score_count.n > n) score_count.n = n;
    }
    super();
  }

  event draw {
    if (timer >= game_over_fade)
      spr::render(std::spr::load("img/game_over.png"), 0, 0);
  }
}
```

draw_numbers.mg

```
// makeshift number drawing. slightly differnet from egg's
using std::spr;

object Draw {
  sprite spr;
  int x; int y; int width; int height;
  int n; int digits;

  event create(int x, int y, int digits) {
    this.x = x; this.y = y; this.digits = digits;
    spr = load("img/numbers.png");
  }

  event draw {
    int div = 1;
    for (int i = 0; i < digits-1; ++i) div *= 10;
    int x = x;
    int n = this.n;
    while (div > 0) {
      int d = (n / div) % 10;
      render_rect(spr, x, y, [8 * d, 0, 8, 8]);
      x += 8;
      div /= 10;
    }
  }
}
```

tetris.ll (generated)

Okay, the generated output is actually extremely verbose since every time we access a member variable, code is generated to access all the member variables (so that codegen can pass around StringMaps of guaranteed llvalues). Presumably all these redundant accesses get optimized out, but for the purposes of the code listing, one of these files adds 70-140 pages, so it's omitted. What we probably could've done better is instead pass around StringMaps of thunks returning llvalues. In any case, the ll files for the demos are included in the demo directory (makergame/demo/egg/egg.ll and makergame/demo/tetris/tetris.ll)

Tests (Steven, Cindy, Yuncheng)

Test runner

```
#!/bin/sh

# Regression testing script for makergame
# Step through a list of files
#  Compile, run, and check the output of each expected-to-work test
#  Compile and check the error of each expected-to-fail test

# Path to the LLVM interpreter
LLI="lli"
#LLI="/usr/local/opt/llvm/bin/lli"

# Path to the LLVM compiler
LLC="llc -relocation-model=pic"

# Path to the C compiler
CC="cc"

# Path to the makergame compiler.  Usually "./makergame.native"
# Try "_build/makergame.native" if ocamlbuild was unable to create a symbolic
link.
makergame="./makergame.native"
#makergame="_build/makergame.native"

LSFML="-lsfml-audio -lsfml-graphics -lsfml-window -lsfml-system -lstdc++"

# Set time limit for all operations
ulimit -t 30

globallog=testall.log
rm -f $globallog
error=0
globalerror=0
```

```
keep=0

Usage() {
      echo "Usage: testall.sh [options] [.mg files]"
      echo "-k    Keep intermediate files"
      echo "-h    Print this help"
      exit 1
}

SignalError() {
      if [ $error -eq 0 ] ; then
    echo "FAILED"
    error=1
      fi
      echo "  $1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile.  Differences, if any, written to
difffile
Compare() {
      generatedfiles="$generatedfiles $3"
      echo diff -b $1 $2 ">" $3 1>&2
      diff -b "$1" "$2" > "$3" 2>&1 || {
    SignalError "$1 differs"
    echo "FAILED $1 differs from $2" 1>&2
      }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
      echo $* 1>&2
      eval $* || {
    SignalError "$1 failed on $*"
    return 1
      }
}

# RunFail <args>
# Report the command, run it, and expect an error
RunFail() {
      echo $* 1>&2
      eval $* && {
    SignalError "failed: $* did not report an error"
    return 1
      }
```

```
      return 0
}

Check() {
      error=0
      basename=`echo $1 | sed 's/.*\\///
                              s/.mg//'`
      reffile=`echo $1 | sed 's/.mg$//'`
      basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

      echo -n "$basename..."

      echo 1>&2
      echo "###### Testing $basename" 1>&2

      generatedfiles=""

      generatedfiles="$generatedfiles ${basename}.ll ${basename}.s $
{basename}.exe ${basename}.out" &&
      Run "$makergame" "<" $1 ">" "${basename}.ll" &&
      Run "$LLC" "${basename}.ll" ">" "${basename}.s" &&
      Run "$CC" "-o" "${basename}.exe" "${basename}.s" "-Lruntime
-ltestergame" &&
      Run "./${basename}.exe" > "${basename}.out" &&
      Compare ${basename}.out ${reffile}.out ${basename}.diff

      # Report the status and clean up the generated files

      if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
      rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
      else
    echo "###### FAILED" 1>&2
    globalerror=$error
      fi
}

CheckFail() {
      error=0
      basename=`echo $1 | sed 's/.*\\///
                              s/.mg//'`
      reffile=`echo $1 | sed 's/.mg$//'`
      basedir="`echo $1 | sed 's/\/[^\/]*$//'`/."

      echo -n "$basename..."
```

```
      echo 1>&2
      echo "###### Testing $basename" 1>&2

      generatedfiles=""

      generatedfiles="$generatedfiles ${basename}.err ${basename}.diff" &&
      RunFail "$makergame" "<" $1 "2>" "${basename}.err" ">>" $globallog &&
      Compare ${basename}.err ${reffile}.err ${basename}.diff

      # Report the status and clean up the generated files

      if [ $error -eq 0 ] ; then
    if [ $keep -eq 0 ] ; then
      rm -f $generatedfiles
    fi
    echo "OK"
    echo "###### SUCCESS" 1>&2
      else
    echo "###### FAILED" 1>&2
    globalerror=$error
      fi
}

while getopts kdpsh c; do
      case $c in
    k) # Keep intermediate files
      keep=1
      ;;
    h) # Help
      Usage
      ;;
      esac
done

shift `expr $OPTIND - 1`

LLIFail() {
  echo "Could not find the LLVM interpreter \"$LLI\"."
  echo "Check your LLVM installation and/or modify the LLI variable in
testall.sh"
  exit 1
}

export OCAMLRUNPARAM=""
export MAKERGAME_PATH="$(pwd)/lib"

which "$LLI" >> $globallog || LLIFail
```

```
if [ ! -f runtime/libtestergame.a ]
then
      echo "Could not find libtestergame.a"
      echo "Try \"make libtestergame.a\""
      exit 1
fi

if [ $# -ge 1 ]
then
      files=$@
else
      files=`ls tests -t | grep .mg`
fi

for file in $files
do
      file="tests/$file"
      case $file in
    *test-*)
      Check $file 2>> $globallog
      ;;
    *fail-*)
      CheckFail $file 2>> $globallog
      ;;
    *)
      echo "unknown file type $file"
      globalerror=1
      ;;
      esac
done

exit $globalerror
```

Test cases

```
==> tests/fail-addasn1.err <==
Fatal error: exception Failure("illegal assign operator int += string in x
+= \"hello\"")

==> tests/fail-addasn1.mg <==
int addAsn(int x)
{
  return x += "hello";
}

object main {
  event create {
```

```
    std::print::i( addAsn(1) );
    std::game::end();
  }
}

==> tests/fail-addasn2.err <==
Fatal error: exception Failure("lvalue 1 expected in 1")

==> tests/fail-addasn2.mg <==
object main {
  event create {
    1 += 2; // LHS is not lvalue
  }
}

==> tests/fail-array1.err <==
Fatal error: exception Failure("lvalue compute(5) expected in compute(5)[3] =
5")

==> tests/fail-array1.mg <==
int compute[10](int x) {
  int ret[10];
  return ret;
}

object main {
  event create {
    compute(5)[3] = 5; // error: cannot assign to element of rvalue
  }
}

==> tests/fail-array2.err <==
Fatal error: exception Failure("array literal [1, 2, true] contains elements
of unequal types int and bool")

==> tests/fail-array2.mg <==
object main {
  event create {
    [1, 2, true];
  }
}

==> tests/fail-array3.err <==
Fatal error: exception Failure("array literal [[1, 2], [3, 4], [5, 6, 7]]
contains elements of unequal types int[2] and int[3]")

==> tests/fail-array3.mg <==
object main {
```

```
  event create {
    [[1, 2], [3, 4], [5, 6, 7]];
  }
}

==> tests/fail-assign1.err <==
Fatal error: exception Failure("illegal assignment int = bool in i = false")

==> tests/fail-assign1.mg <==
object main {
  event create {
    int i;
    bool b;

    i = 42;
    i = 10;
    b = true;
    b = false;
    i = false; /* Fail: assigning a bool to an integer */
  }
}

==> tests/fail-assign2.err <==
Fatal error: exception Failure("illegal assignment bool = int in b = 48")

==> tests/fail-assign2.mg <==
object main {
  event create {
    int i;
    bool b;

    b = 48; /* Fail: assigning an integer to a bool */
  }
}

==> tests/fail-assign3.err <==
Fatal error: exception Failure("illegal assignment int = void in i =
myvoid()")

==> tests/fail-assign3.mg <==
void myvoid()
{
  return;
}

object main {
  event create {
    int i;
```

```
    i = myvoid(); /* Fail: assigning a void to an integer */
  }
}

==> tests/fail-assign4.err <==
Fatal error: exception Failure("lvalue create helper expected in create helper
= x")

==> tests/fail-assign4.mg <==
object helper { }
object main {
  event create {
    helper x;
    create helper = x; // not an lvalue
  }
}

==> tests/fail-assign5.err <==
Fatal error: exception Failure("lvalue compute() expected in compute() = 5")

==> tests/fail-assign5.mg <==
int compute() { return 3; }
object main {
  event create {
    compute() = 5; // not an lvalue
  }
}

==> tests/fail-assign6.err <==
Fatal error: exception Failure("lvalue 3 + 4 expected in 3 + 4 = 7")

==> tests/fail-assign6.mg <==
object main {
  event create {
    3 + 4 = 7; // not an lvalue
  }
}

==> tests/fail-break1.err <==
Fatal error: exception Failure("cannot break in top level block of
::main::create")

==> tests/fail-break1.mg <==
object main {
  event create {
    break; // error: cannot break in outermost block
  }
```

```
}

==> tests/fail-break2.err <==
Fatal error: exception Failure("cannot break in top level block of
::main::step")

==> tests/fail-break2.mg <==
object main { event step { break; } }

==> tests/fail-compare2.err <==
Fatal error: exception Failure("illegal binary operator aide == helper in a ==
h")

==> tests/fail-compare2.mg <==
object helper { }
object aide { }

object main {
  event create {
    helper h;
    aide a;
    std::print::b(a == h); // cannot compare unrelated objects
    std::game::end();
  }
}

==> tests/fail-compare.err <==
Fatal error: exception Failure("illegal binary operator sprite == sprite in a
== a")

==> tests/fail-compare.mg <==
object main {
  sprite a;
  event create {
    bool x;
    x = (a == a);
  }
}

==> tests/fail-constexpr2.err <==
Fatal error: exception Failure("illegal assignment to global variable: int x =
y")

==> tests/fail-constexpr2.mg <==
int y = 3;
int x = y; // forbidden (for now)
object main { }
```

```
==> tests/fail-constexpr.err <==
Fatal error: exception Failure("illegal assignment to global variable: int x =
3 + 5")

==> tests/fail-constexpr.mg <==
int x = 3 + 5; // forbidden (for now)
object main { }

==> tests/fail-create1.err <==
Fatal error: exception Parsing.Parse_error

==> tests/fail-create1.mg <==
object main {
  event create { create 5; } /* error: can't create non-object type */
}

==> tests/fail-create2.err <==
Fatal error: exception Failure("illegal assignment int = main in x = create
main")

==> tests/fail-create2.mg <==
object main {
  event create {
    int x;
    x = create main; /* error: must assign to object type */
  }
}

==> tests/fail-create3.err <==
Fatal error: exception Failure("unrecognized function super")

==> tests/fail-create3.mg <==
object parent {
  event create(int x) { super(); }
}

object main { event create { super(); } } // main has no parent


==> tests/fail-decl.err <==
Fatal error: exception Failure("duplicate local 'x' in single block of
::main::step")

==> tests/fail-decl.mg <==
object main {
  event step {
    int x;
    int x; // error: redeclaration in same block
```

```
  }
}

==> tests/fail-destroy1.err <==
Fatal error: exception Failure("cannot destroy non-object")

==> tests/fail-destroy1.mg <==
object main {
  event create { destroy 5; } /* error: can't destroy non-object ref */
}

==> tests/fail-destroy2.err <==
Fatal error: exception Failure("illegal assignment int = void in x = destroy
m")

==> tests/fail-destroy2.mg <==
main m;
object main {
  event create {
    int x;
    x = destroy m; /* error: cannot assign result of destroy */
  }
}

==> tests/fail-expr1.err <==
Fatal error: exception Failure("illegal binary operator bool + int in d + a")

==> tests/fail-expr1.mg <==
int a;
bool b;

void foo(int c, bool d)
{
  int dd;
  bool e;
  a + c;
  c - a;
  a * 3;
  c / 2;
  d + a; /* Error: bool + int */
}

object main { }

==> tests/fail-expr2.err <==
Fatal error: exception Failure("illegal binary operator bool + int in b + a")

==> tests/fail-expr2.mg <==
```

```
int a;
bool b;

void foo(int c, bool d)
{
  int d;
  bool e;
  b + a; /* Error: bool + int */
}

object main { }

==> tests/fail-files1.err <==
Fatal error: exception Failure("unable to open file 'nonexistent12345555.mg'")

==> tests/fail-files1.mg <==
// nonexistent file
namespace c = open "nonexistent12345555.mg";

object main { }

==> tests/fail-files2.err <==
Fatal error: exception Failure("failed to parse file test-helpers/bad-
parse.mg")

==> tests/fail-files2.mg <==
// file does not parse correctly
namespace c = open "test-helpers/bad-parse.mg";

object main { }

==> tests/fail-files3.err <==
Fatal error: exception Failure("return gives string expected int
in \"hello\"")

==> tests/fail-files3.mg <==
// file fails semantic check
namespace c = open "test-helpers/bad-semant.mg";

object main { }

==> tests/fail-files4.err <==
Fatal error: exception Failure("attempted access to private namespace a in
resolving n::a")

==> tests/fail-files4.mg <==
namespace n = open "test-helpers/private.mg";
```

```
object main {
  event create {
    n::a::x = 0; // error: n::a is private
  }
}

==> tests/fail-files5.err <==
Fatal error: exception Failure("attempted access to private namespace std in
resolving n::std::print")

==> tests/fail-files5.mg <==
namespace n {
  // std is imported into every namespace, so following is valid
  void do_something() { std::print::i(3); }
}

object main {
  event create {
    std::print::i(3);
    n::std::print::i(5); // error: n::std is private
  }
}

==> tests/fail-files6.err <==
Fatal error: exception Failure("circular file dependency with recursive.mg")

==> tests/fail-files6.mg <==
// update: includes itself so it fails
// old: contains namespace definition that involves itself
namespace file = open "../test-helpers/recursive.mg";

object main { }

==> tests/fail-files7.err <==
Fatal error: exception Failure("circular file dependency with circular.mg")

==> tests/fail-files7.mg <==
// used to be able to have files include themselves. now forbidden.
namespace c = open "test-helpers/circular.mg"; // includes itself

object main {
  event create {
    c::x = 12;
    std::print::i(c::c::x);
    std::print::i(c::c::c::x);
    std::print::i(c::c::c::c::x);
    std::game::end();
  }
```

```
}

==> tests/fail-for1.err <==
Fatal error: exception Failure("undeclared identifier j")

==> tests/fail-for1.mg <==
object main {
  event create {
    int i;
    for ( ; true ; ) {} /* OK: Forever */

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

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

==> tests/fail-for2.err <==
Fatal error: exception Failure("undeclared identifier j")

==> tests/fail-for2.mg <==
object main {
  event create {
    int i;

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

==> tests/fail-for3.err <==
Fatal error: exception Failure("expected Boolean expression in i")

==> tests/fail-for3.mg <==
object main {
event create {
  int i;

  for (i = 0; i ; i = i + 1) {} /* i is an integer, not Boolean */
}
}

==> tests/fail-for4.err <==
Fatal error: exception Failure("undeclared identifier j")

==> tests/fail-for4.mg <==
object main {
```

```
event create
{
  int i;

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

==> tests/fail-for5.err <==
Fatal error: exception Failure("unrecognized function foo")

==> tests/fail-for5.mg <==
object main {
event create
{
  int i;

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

==> tests/fail-fordef1.err <==
Fatal error: exception Parsing.Parse_error

==> tests/fail-fordef1.mg <==
object main {
  event create {
      for (return; i < 5 ; i++) { }
  }
}

==> tests/fail-fordef2.err <==
Fatal error: exception Failure("undeclared identifier i")

==> tests/fail-fordef2.mg <==
object main {
event create
{
  for (int i = 1; i < 5 ; i = i + 1) {
    std::print::i(i);
  }
  std::print::i(i); // out of scope
  return;
}
}
```

```
==> tests/fail-foreach1.err <==
Fatal error: exception Failure("can only use foreach loop on objects in int
x")

==> tests/fail-foreach1.mg <==
object main {
  event create {
    foreach (int x) { std::print::i(x); } /* error: can only loop over objs */
  }
}

==> tests/fail-foreach2.err <==
Fatal error: exception Failure("unrecognized game object ::asdf")

==> tests/fail-foreach2.mg <==
object main {
  event create {
    foreach (asdf x) { std::print::i(3); } /* error: obj nonexistent */
  }
}

==> tests/fail-func1.err <==
Fatal error: exception Failure("duplicate function ::bar")

==> tests/fail-func1.mg <==
int foo() {}

int bar() {}

int baz() {}

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

object main { }

==> tests/fail-func2.err <==
Fatal error: exception Failure("duplicate formal a in ::bar")

==> tests/fail-func2.mg <==
int foo(int a, bool b, int c) { }

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

object main { }

==> tests/fail-func3.err <==
Fatal error: exception Failure("illegal void formal b in ::bar")
```

```
==> tests/fail-func3.mg <==
int foo(int a, bool b, int c) { }

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

object main { }

==> tests/fail-func4.err <==
Fatal error: exception Failure("duplicate function ::print")

==> tests/fail-func5.err <==
Fatal error: exception Failure("illegal void local b in ::bar")

==> tests/fail-func5.mg <==
int foo() {}

int bar() {
  int a;
  void b; /* Error: illegal void local b */
  bool c;

  return;
}

object main { }

==> tests/fail-func6.err <==
Fatal error: exception Failure("expecting 2 arguments in foo(42)")

==> tests/fail-func6.mg <==
void foo(int a, bool b)
{
}

object main {
event create
{
  foo(42, true);
  foo(42); /* Wrong number of arguments */
}
}

==> tests/fail-func7.err <==
Fatal error: exception Failure("expecting 2 arguments in foo(42, true,
false)")

==> tests/fail-func7.mg <==
```

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

object main {
event create
{
  foo(42, true);
  foo(42, true, false); /* Wrong number of arguments */
}
}
```

```
==> tests/fail-func8.err <==
Fatal error: exception Failure("illegal actual argument found void expected
bool in bar()")
```

```
==> tests/fail-func8.mg <==
void foo(int a, bool b)
{
}

void bar()
{
}

object main {
event create
{
  foo(42, true);
  foo(42, bar()); /* int and void, not int and bool */
}
}
```

```
==> tests/fail-func9.err <==
Fatal error: exception Failure("illegal actual argument found int expected
bool in 42")
```

```
==> tests/fail-func9.mg <==
void foo(int a, bool b)
{
}

object main {
event create
{
  foo(42, true);
  foo(42, 42); /* Fail: int, not bool */
}
```

```
}

==> tests/fail-global1.err <==
Fatal error: exception Failure("illegal void global ::a")

==> tests/fail-global1.mg <==
int c;
bool b;
void a; /* global variables should not be void */


object main { }

==> tests/fail-global2.err <==
Fatal error: exception Failure("duplicate global ::b")

==> tests/fail-global2.mg <==
int b;
bool c;
int a;
int b; /* Duplicate global variable */

object main { }

==> tests/fail-if1.err <==
Fatal error: exception Failure("expected Boolean expression in 42")

==> tests/fail-if1.mg <==
object main {
event create
{
  if (true) {}
  if (false) {} else {}
  if (42) {} /* Error: non-bool predicate */
}
}

==> tests/fail-if2.err <==
Fatal error: exception Failure("undeclared identifier foo")

==> tests/fail-if2.mg <==
object main {
event create
{
  if (true) {
    foo; /* Error: undeclared variable */
  }
}
```

```
}

==> tests/fail-if3.err <==
Fatal error: exception Failure("undeclared identifier bar")

==> tests/fail-if3.mg <==
object main {
event create
{
  if (true) {
    42;
  } else {
    bar; /* Error: undeclared variable */
  }
}
}

==> tests/fail-inc-dec1.err <==
Fatal error: exception Failure("lvalue 42 expected in ++42")

==> tests/fail-inc-dec1.mg <==
object main {
  event create {
    std::print::i(++42);
    std::print::i(--42);
    std::game::end();
  }
}

==> tests/fail-inheritance2.err <==
Fatal error: exception Failure("inheritance cycle with main")

==> tests/fail-inheritance2.mg <==
object main : main { event create { } }

==> tests/fail-inheritance3.err <==
Fatal error: exception Failure("expecting 1 arguments in create child")

==> tests/fail-inheritance3.mg <==
object parent { event create(int x)  { } }
object child : parent { }

object main {
  event create { create child; } // error: requires argument
}

==> tests/fail-inheritance.err <==
Fatal error: exception Failure("inheritance cycle with A")
```

```
==> tests/fail-inheritance.mg <==
object main { event create { } }

object A : B { }
object B : A { }

==> tests/fail-minusasn1.err <==
Fatal error: exception Failure("illegal assign operator int -= string in x -=
\"hello\"")

==> tests/fail-minusasn1.mg <==
int minusAsn(int x)
{
  return x -= "hello";
}

object main {
event create
{
  std::print::i( minusAsn(1) );

  std::game::end();
}
}

==> tests/fail-modulo.err <==
Fatal error: exception Failure("illegal binary operator bool % int in true %
2")

==> tests/fail-modulo.mg <==
object main { event create { true % 2; } }

==> tests/fail-namespace2.err <==
Fatal error: exception Failure("undeclared identifier x")

==> tests/fail-namespace2.mg <==
int x;
namespace sp {
  void modify() { x = 3; } // error: cannot access outside namespace
}

object main { }

==> tests/fail-namespace3.err <==
Fatal error: exception Failure("duplicate namespace ::sp")

==> tests/fail-namespace3.mg <==
```

```
namespace sp { }
namespace sp { } // error: duplicate namespace

object main { }

==> tests/fail-namespace4.err <==
Fatal error: exception Failure("duplicate namespace sp::a")

==> tests/fail-namespace4.mg <==
namespace sp { namespace a { } namespace a { } } // error: duplicate namespace

object main { }

==> tests/fail-namespace5.err <==
Fatal error: exception Failure("unrecognized namespace a encountered when
resolving a")

==> tests/fail-namespace5.mg <==
namespace b = a; // reference to undefined namespace

object main { }

==> tests/fail-namespace.err <==
Fatal error: exception Failure("cannot shadow or overwrite identifier 'this'
in sp::hello members")

==> tests/fail-namespace.mg <==
namespace sp {
  object hello {
    int this; /* error in namespace */
  }
}

object main {
  event create {
    std::print::s("hey");
    std::game::end();
  }
}

==> tests/fail-nomain.err <==
Fatal error: exception Failure("main game object must be defined")

==> tests/fail-nomain.mg <==

==> tests/fail-ns-alias1.err <==
Fatal error: exception Failure("unrecognized namespace a in a")
```

```
==> tests/fail-ns-alias2.err <==
Fatal error: exception Failure("namespace A never resolves")

==> tests/fail-ns-alias2.mg <==
// namespace alias cycle
namespace A = B;
namespace B = A;

object main { }

==> tests/fail-ns-alias3.err <==
Fatal error: exception Failure("unrecognized namespace c encountered when
resolving a::c")

==> tests/fail-ns-alias3.mg <==
// invalid alias to inner namespace
namespace a { }
namespace B = a::c;

object main { }

==> tests/fail-ns-std.err <==
Fatal error: exception Failure("duplicate namespace ::std")

==> tests/fail-ns-std.mg <==
int foo() {}

void bar() {}

namespace std {} /* Should not be able to define std */

void baz() {}

object main { }

==> tests/fail-objdecl.err <==
Fatal error: exception Failure("unrecognized game object
::somenonexistentobject")

==> tests/fail-objdecl.mg <==
object main {
  event create {
    somenonexistentobject a;
  }
}

==> tests/fail-objfn1.err <==
Fatal error: exception Failure("duplicate methods same in main")
```

```
==> tests/fail-objfn1.mg <==
// error: duplicate fns
object main {
  void same() { }
  void same() { }
}

==> tests/fail-return1.err <==
Fatal error: exception Failure("return gives bool expected int in true")

==> tests/fail-return1.mg <==
int hello() {
  return true; /* Should return int */
}

object main { }

==> tests/fail-return2.err <==
Fatal error: exception Failure("return gives int expected void in 42")

==> tests/fail-return2.mg <==
void foo()
{
  if (true) return 42; /* Should return void */
  else return;
}

object main { }

==> tests/fail-scope2.err <==
Fatal error: exception Failure("undeclared identifier y")

==> tests/fail-scope2.mg <==
object main {
event create {
  int x;
  x = 3;
  {
    int y;
    y = 4;
  }
  std::print::i(x);
  std::print::i(y); /* error: y not in outer scope */
}
}

==> tests/fail-scope.err <==
```

```
Fatal error: exception Failure("undeclared identifier x")

==> tests/fail-scope.mg <==
object main {
event create {
  x = 3; /* error: variable used before defined */
  int x;
}
}

==> tests/fail-string1.err <==
Fatal error: exception Failure("illegal assignment int = string in c
= \"hello\"")

==> tests/fail-string1.mg <==
object main {
event create {
  int c;
  c = "hello";
}
}

==> tests/fail-string2.err <==
Fatal error: exception Failure("illegal assignment string = int in c = 4")

==> tests/fail-string2.mg <==
object main {
event create {
  string c;
  c = 4;
}
}

==> tests/fail-string3.err <==
Fatal error: exception Failure("illegal actual argument found int expected
string in 4")

==> tests/fail-string3.mg <==
object main {
event create {
  std::print::s(4);
}
}

==> tests/fail-string4.err <==
Fatal error: exception Failure("illegal actual argument found string expected
int in \"hello\"")
```

```
==> tests/fail-string4.mg <==
object main {
event create {
  std::print::i("hello");
}
}

==> tests/fail-string5.err <==
Fatal error: exception Failure("unsupported escape character \\w")

==> tests/fail-string5.mg <==
object main {
event create {
  std::print::s("hello\world");
}
}

==> tests/fail-string6.err <==
Fatal error: exception Failure("illegal character \\'")

==> tests/fail-string6.mg <==
object main {
event create {
  std::print::s('yesterday');
}
}

==> tests/fail-string7.err <==
Fatal error: exception Parsing.Parse_error

==> tests/fail-string7.mg <==
object main {
event create {
  std::print::s("is "this" wrong?");
}
}

==> tests/fail-super.err <==
Fatal error: exception Failure("expecting 0 arguments in super(3)")

==> tests/fail-super.mg <==
object child : parent {
  event step { super(3); } // super for step takes no arguments
}

object parent {
  event create(int x) { std::print::i(x); }
}
```

```
object main { }

==> tests/fail-this1.err <==
Fatal error: exception Failure("'this' cannot be assigned in 'this = create
main'")

==> tests/fail-this1.mg <==
object main {
  event create {
    this = create main; /* this cannot be assigned */
  }
}

==> tests/fail-this2.err <==
Fatal error: exception Failure("cannot shadow or overwrite identifier 'this'
in ::main::create")

==> tests/fail-this2.mg <==
object main {
  event create {
    int this; /* error: this cannot be declared */

    this = 3;
  }
}

==> tests/fail-this3.err <==
Fatal error: exception Failure("cannot shadow or overwrite identifier 'this'
in formals of ::compute")

==> tests/fail-this3.mg <==
int compute(int this) { return 3; } /* error: this cannot be an argument name
*/

object main { }

==> tests/fail-this4.err <==
Fatal error: exception Failure("cannot shadow or overwrite identifier 'this'
in  globals")

==> tests/fail-this4.mg <==
int this; /* error: this cannot be a global variable name */

object main { }

==> tests/fail-this5.err <==
```

```
Fatal error: exception Failure("cannot shadow or overwrite identifier 'this'
in gameobj declarations")

==> tests/fail-this5.mg <==
object this { } /* error: this cannot be a gameobj name */

object main { }

==> tests/fail-this6.err <==
Fatal error: exception Failure("cannot shadow or overwrite identifier 'this'
in ::main members")

==> tests/fail-this6.mg <==
object main {
  int this; /* error: this cannot be a gameobj member */
}

==> tests/fail-vdef1.err <==
Fatal error: exception Failure("undeclared identifier x")

==> tests/fail-vdef1.mg <==
object main {
  event create
  {
    int x = x + 4;
  }
}

==> tests/fail-vdef2.err <==
Fatal error: exception Failure("illegal assignment int = string in \"hello\"")

==> tests/fail-vdef2.mg <==
object main {
  event create
  {
    int a = "hello";
  }
}

==> tests/fail-while1.err <==
Fatal error: exception Failure("expected Boolean expression in 42")

==> tests/fail-while1.mg <==
object main {
event create
{
  int i;
```

```
  while (true) {
    i = i + 1;
  }

  while (42) { /* Should be boolean */
    i = i + 1;
  }

}
}

==> tests/fail-while2.err <==
Fatal error: exception Failure("unrecognized function foo")

==> tests/fail-while2.mg <==
object main {
event create
{
  int i;

  while (true) {
    i = i + 1;
  }

  while (true) {
    foo(); /* foo undefined */
  }

}
}

==> tests/test-add1.mg <==
int add(int x, int y)
{
  return x + y;
}

object main {
event create
{
  std::print::i( add(17, 25) );

  std::game::end();
}
}

==> tests/test-add1.out <==
42
```

```
==> tests/test-addasn1.mg <==
int addAsn(int x)
{
  return x += 2;
}

object main {
event create
{
  std::print::i( addAsn(1) );

  std::game::end();
}
}

==> tests/test-addasn1.out <==
3

==> tests/test-addasn2.mg <==
object main {
event create
{
  float x;
  x = 3.0;
  x += 3.0;
  std::print::f( x );

  std::game::end();
}
}

==> tests/test-addasn2.out <==
6.000000

==> tests/test-addasn3.mg <==
int addAsn(int x)
{
  int y;
  y = ((x += 2) = 3);
  return y;
}

object main {
event create
{
  std::print::i( addAsn(1) );
```

```
  std::game::end();
}
}

==> tests/test-addasn3.out <==
3

==> tests/test-addasn4.mg <==
object main {
event create
{
  int x;
  x = 0;
  std::print::i((x += 2) += 2);
  std::print::i(x);
  std::game::end();
}
}

==> tests/test-addasn4.out <==
4
4

==> tests/test-addasn5.mg <==
object main {
  event create
  {
    int x = 5;
    x -= 2.5; // turns to 5 - 2 = 3

    float y = 10.0;
    y -= 3; // turns to 10.0 - 3.0 = 7.0
    std::print::i(x);
    std::print::f(y);
    std::game::end();
  }
}

==> tests/test-addasn5.out <==
3
7.000000

==> tests/test-arith1.mg <==
object main {
event create
{
  std::print::i(39 + 3);
  std::game::end();
```

```
}
}

==> tests/test-arith1.out <==
42

==> tests/test-arith2.mg <==
object main {
event create
{
  std::print::i(1 + 2 * 3 + 4);
  std::game::end();
}
}

==> tests/test-arith2.out <==
11

==> tests/test-arith3.mg <==
int foo(int a)
{
  return a;
}

object main {
event create
{
  int a;
  a = 42;
  a = a + 5;
  std::print::i(a);
  std::game::end();
}
}

==> tests/test-arith3.out <==
47

==> tests/test-array1.mg <==
object main {
  event create {
    int j[10];
    int i;
    for (i = 0; i < 10; ++i)
      j[i] = i;
    std::print::i(j[3]);
    std::game::end();
  }
```

```
}

==> tests/test-array1.out <==
3

==> tests/test-array2.mg <==
// array returns
int make_ten_of[10](int x) {
  int ret[10];
  int i;
  for (i = 0; i < 10; ++i)
    ret[i] = x;
  return ret;
}

object main {
  event create {
    int i = 3;
    int j[10] = make_ten_of(5);
    std::print::i(j[i]);
    std::game::end();
  }
}

==> tests/test-array2.out <==
5

==> tests/test-array3.mg <==
// array arguments
int sum(int x[3]) {
  int sum = 0;
  int i;
  for (i = 0; i < 3; ++i)
    sum += x[i];
  return sum;
}

object main {
  event create {
    int i[3];
    i[0] = 1;
    i[1] = 2;
    i[2] = 3;
    std::print::i(sum(i));
    std::game::end();
  }
}
```

```
==> tests/test-array3.out <==
6

==> tests/test-array4.mg <==
// multidimensional arrays
object main {
  event create {
    int x[5][2] = [[00, 01], [10, 11], [20, 21], [30, 31], [40, 41]];
    std::print::i(x[3][1]);
    std::game::end();
  }
}

==> tests/test-array4.out <==
31

==> tests/test-array5.mg <==
object main {
  event create {
    int x[5] = [1, 2, 3, 4, 5];
    int i; int sum = 0;
    for (i = 0; i < 5; ++i)
      sum += x[i];
    std::print::i(sum);
    std::game::end();
  }
}

==> tests/test-array5.out <==
15

==> tests/test-array6.mg <==
// multidimensional arrays
// non-const expressions in literals
object main {
  event create {
    int y = 3;
    int x[3][3] = [[1, 2, y], [4, 5, 6], [7, 8, 9]];
    int i; int j;
    for (i = 0; i < 3; ++i)
      for (j = 0; j < 3; ++j)
        std::print::i(x[i][j]);
    std::game::end();
  }
}

==> tests/test-array6.out <==
1
```

```
2
3
4
5
6
7
8
9

==> tests/test-array7.mg <==
int array[2][3] = [[1,2,3],[4,5,6]];
int compute[3]() { return array[0]; }
object main {
  event create {
    int arr[3] = compute();
    std::print::i(arr[2]);
    std::game::end();
  }
}

==> tests/test-array7.out <==
3

==> tests/test-assign1.mg <==
object helper {
  int x;
  event create { x = 5; }
}

helper make() { return create helper; }

object main {
  event create {
    // member variables of anything resolving to an object can be assigned
    // not just chains of identifiers

    make().x = 3;
    foreach (helper h) { std::print::i(h.x); destroy h; }

    (create helper).x = 4;
    foreach (helper h) { std::print::i(h.x); destroy h; }

    std::print::i(make().x);

    helper h;
    (h = create helper).x = 4;
    std::print::i(h.x);
```

```
    // assignments are also lvalues
    (h.x = 5) = 6;
    std::print::i(h.x);

    std::game::end();
  }
}

==> tests/test-assign1.out <==
3
4
5
4
6

==> tests/test-break1.mg <==
object main {
event create {
  int i;
  i = 0;
  while (i < 5) {
    if (i == 2) { break; } // breaks out of loop
    std::print::i(i);
    i = i + 1;
  }
  std::game::end();
}
}

==> tests/test-break1.out <==
0
1

==> tests/test-break2.mg <==
object main {
event create {
  int i;
  for (i = 0; i < 5; i = i + 1) {
    if (i == 2) { break; } // breaks out of loop
    std::print::i(i);
  }
  std::game::end();
}
}

==> tests/test-break2.out <==
0
1
```

```
==> tests/test-break3.mg <==

object helper { }
object main {
  event create {
    create helper;
    create helper;
    create helper;
    create helper;
    int i;
    i = 0;
    foreach (helper h) {
      if (i == 2) break;
      std::print::i(i);
      i = i + 1;
    }
    std::game::end();
  }
}

==> tests/test-break3.out <==
0
1

==> tests/test-compare2.mg <==
// comparisons with inheritance chains
object helper : parent { }
object parent { }

object main {
  event create {
    helper h = create helper;
    parent P = create parent;
    parent H = h;
    std::print::b(h == h);
    std::print::b(H == h);
    std::print::b(h == H);
    std::print::b(h == P);
    std::print::b(P == h);
    std::game::end();
  }
}

==> tests/test-compare2.out <==
true
true
true
```

```
false
false

==> tests/test-compare3.mg <==
object main {
  event create {
    std::print::b(2 == 1.0);
    std::game::end();
  }
}

==> tests/test-compare3.out <==
false

==> tests/test-compare4.mg <==
object helper { }
object aide { }

object main {
  event create {
    helper h = create helper;
    helper g = none;
    std::print::b(h == g);
    std::print::b(h == none);
    std::print::b(g == none);

    aide a = none;
    std::print::b(a == none);
    std::game::end();
  }
}

==> tests/test-compare4.out <==
false
false
true
true

==> tests/test-compare.mg <==
object helper { }
object main {
  event create {
    for (int i = 0; i < 5; ++i) create helper;
    int i = 0;
    // each of the helpers is identical only to itself
    foreach (helper h)
      foreach (helper g)
        if (h == g) ++i;
```

```
    std::print::i(i);
    std::game::end();
  }
}

==> tests/test-compare.out <==
5

==> tests/test-constexpr2.mg <==
int x[3][5] = [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]];

object main {
  event create {
    for (int i = 0; i < 3; ++i)
      for (int j = 0; j < 5; ++j)
        std::print::i(x[i][j]);
    std::game::end();
  }
}

==> tests/test-constexpr2.out <==
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

==> tests/test-constexpr.mg <==
int x = 3;

object main {
  event create { std::print::i(x); std::game::end(); }
}

==> tests/test-constexpr.out <==
3

==> tests/test-create2.mg <==
```

```
object helper {
  // should run before main's second step
  event step { std::print::s("helper"); }
}

object aide {
  // should run before main's second step
  event step { std::print::s("aide"); }
}

object main {
  bool created;
  event create { this.created = false; create aide; }
  event step {
    std::print::s("main");
    if (!this.created) {
      create helper;
      create helper;
      this.created = true;
    }
    else {
      create aide; // causes an 'aide' call right before the end
      std::game::end();
    }
  }
}


==> tests/test-create2.out <==
main
aide
helper
helper
main
aide
helper
helper
aide

==> tests/test-create3.mg <==
// constructors with parameters
object helper {
  int x;
  event create(int x) {
    this.x = x;
  }
}
```

```
object main {
  event create {
    helper h;
    h = create helper(4);
    std::print::i(h.x);
    std::game::end();
  }
}


==> tests/test-create3.out <==
4

==> tests/test-create4.mg <==
object child : parent {
  event create(int x) { super(x+1); std::print::i(x); }
}

object parent {
  event create(int x) { std::print::i(x); }
}

object main {
  event create {
    create child(3);
    std::game::end();
  }
}


==> tests/test-create4.out <==
4
3

==> tests/test-create5.mg <==
object child : parent {
  event create(int x) { super(); std::print::i(x); }
}

object parent {
  event create() { std::print::i(4); }
}

object main {
  event create {
    create child(3);
    std::game::end();
```

```
  }
}


==> tests/test-create5.out <==
4
3

==> tests/test-create.mg <==

object helper {
  int y;
  event create { this.y = 3; }
}

object main {
  int y;
  helper h;
  event create { this.y = 4; this.h = create helper; }
  event step {
    int y;
    y = 5;
    std::print::i(y);
    std::print::i(this.y);
    std::print::i(this.h.y);
    std::game::end();
  }
}

==> tests/test-create.out <==
5
4
3

==> tests/test-destroy1.mg <==

object helper {
  int y;
  event create { this.y = 3; }
  event destroy { std::print::i(this.y); std::game::end(); }
}

object main {
  int y;
  helper h;
  event create { this.h = create helper; }
  event step {
    destroy this.h;
```

```
  }
}

==> tests/test-destroy1.out <==
3

==> tests/test-destroy2.mg <==

object helper {
  event create { destroy this; }
}

object main {
  event create {
    create helper;
    int i;
    i = 0; /* count # helpers */
    foreach (helper h) { i = i + 1; }
    std::print::i(i);
    std::game::end();
  }
}

==> tests/test-destroy2.out <==
0

==> tests/test-destroy3.mg <==

object helper { }

object main {
  event create {
    int i;
    for (i = 0; i < 5; i = i + 1) create helper;
    /* outer loop should not iterate over things that were destroyed */
    foreach (helper x) {
      std::print::s("outer");
      foreach (helper y) {
        std::print::s("inner");
        destroy y;
      }
    }
    std::game::end();
  }
}

==> tests/test-destroy3.out <==
outer
```

```
inner
inner
inner
inner
inner
```

==> tests/test-destroy4.mg <==

```
object helper {
  int i;
  event create { this.i = 3; }
}

object main {
  event create {
    helper h;
    h = create helper;
    destroy h;
    /* destroyed objects are immediately invisible to loops */
    foreach (helper h) { std::print::s("still exists!"); }
    /* refs to destroyed objects still work until at least end of event */
    std::print::i(h.i);
    std::game::end();
  }
}
```

==> tests/test-destroy4.out <==

```
3
```

==> tests/test-destroy5.mg <==

```
object helper { }

object main {
  int j;
  event create {
    this.j = 0;
    int i;
    for (i = 0; i < 5; i = i + 1) create helper;
    /* outer loop should not iterate over things that were destroyed */
    foreach (helper x) {
      std::print::s("outer");
      foreach (helper y) {
        std::print::s("inner");
        destroy x; // 1st helper destroyed 5 extra times; should not break
        destroy y;
      }
    }
```

```
  }
  event step {
    if (this.j > 20) std::game::end();
    this.j = this.j + 1;
  }
}

==> tests/test-destroy5.out <==
outer
inner
inner
inner
inner
inner

==> tests/test-destroy6.mg <==

int i;

object main {
  event create {
    if (i < 5) {
      destroy this;
      i += 1;
      std::print::i(i);
      create main;
    }
    else std::game::end();
  }
}

==> tests/test-destroy6.out <==
1
2
3
4
5

==> tests/test-destroy7.mg <==
object parent {
  void detonate() { destroy this; }
}

object child : parent { }

object main {
  event create {
    for (int i = 0; i < 10; ++i) create child;
```

```
    for (int i = 0; i < 10; ++i) create parent;
    int i = 0;
    foreach (child c) c.detonate();
    foreach (parent c) ++i;
    std::print::i(i);
    i = 0;
    foreach (child c) ++i;
    std::print::i(i);
    std::game::end();
  }
}

==> tests/test-destroy7.out <==
10
0

==> tests/test-destroy8.mg <==
object parent {
  void detonate() { destroy this; }
}

object child : parent { }

object main {
  int j;
  event create { j = 0; }
  event step {
    // Create 10 children, then destroy them using
    // handles to the parent
    for (int i = 0; i < 10; ++i) create child;
    for (int i = 0; i < 10; ++i) create parent;
    int i = 0;
    foreach (child c) c.detonate();
    foreach (parent c) ++i;
    std::print::i(i);

    // run this thing for 6 steps
    ++j;
    if (j >= 6)
      std::game::end();
  }
}

==> tests/test-destroy8.out <==
10
20
30
40
```

```
50
60

==> tests/test-destroy9.mg <==
// sort of minimal example of bug in past
namespace game = std::game;

object A {
  event create { std::print::s("create A"); }
  event step { std::print::s("step A"); }
  event destroy { std::print::s("destroy A"); }
}
object main {
  int x;
  event create { x = 0; }
  event step { ++x; if (x > 5) { destroy this; create maind; create A; } }
}
object maind {
  event create {
    std::print::s("create maind");
    // create A;
  }

  event step { create gameover; }

  event destroy { std::print::s("destroy maind"); }
}

object gameover {
  int i;
  event create {
    std::print::s("create gameover");
    i = 0;
    foreach (object r) { if (r != this) { destroy r; } }
    foreach (object r) { std::print::s("loop alive"); }
  }

  event step {
    ++i;
    foreach (object r) { std::print::s("step alive"); }
    if (i > 5)
      std::game::end();
  }
}

==> tests/test-destroy9.out <==
create maind
create A
```

```
create gameover
destroy maind
destroy A
loop alive
step alive
step alive
step alive
step alive
step alive
step alive
step alive

==> tests/test-fib.mg <==
int fib(int x)
{
  if (x < 2) return 1;
  return fib(x-1) + fib(x-2);
}

object main {
event create
{
  std::print::i(fib(0));
  std::print::i(fib(1));
  std::print::i(fib(2));
  std::print::i(fib(3));
  std::print::i(fib(4));
  std::print::i(fib(5));
  std::game::end();
  return;
}
}

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

==> tests/test-files1.mg <==

// load a file
namespace n = open "test-helpers/simple.mg";

object main {
  event create {
    // set a value defined in that file
```

```
    n::x = 3;
    std::print::i(n::x);
    std::game::end();
  }
}

==> tests/test-files1.out <==
3

==> tests/test-files2.mg <==

namespace one = open "test-helpers/simple.mg";

namespace two = open "test-helpers/simple2.mg";

object main {
  event create {
    // set a value defined in that file
    one::x = 1;
    two::x = 2;
    std::print::i(one::x);
    std::print::i(two::one::x); // two::one refers to the same as one
    std::print::i(two::x);
    std::game::end();
  }
}

==> tests/test-files2.out <==
1
1
2

==> tests/test-files3.out <==
12
12
12

==> tests/test-files4.mg <==
namespace n {
  private namespace p { int x; }
  namespace q = p;
}

object main {
  event create {
    n::q::x = 3; // x is accessible through q
    std::print::i(n::q::x);
    std::game::end();
```

```
  }
}

==> tests/test-files4.out <==
3

==> tests/test-float1.mg <==
object main {
event create {
  float f1;
  f1 = 3.5;
  float f2;
  f2 = 2.5;
  float sum;
  sum = f1 + f2;
  std::print::f(sum);
  std::game::end();
}
}

==> tests/test-float1.out <==
6.000000

==> tests/test-float2.mg <==

object main {
event create
{
  std::print::f(3.5+2.5);
  std::game::end();
}
}

==> tests/test-float2.out <==
6.000000

==> tests/test-for1.mg <==
object main {
event create
{
  int i;
  for (i = 0 ; i < 5 ; i = i + 1) {
    std::print::i(i);
  }
  std::print::i(42);
  std::game::end();
  return;
}
```

```
}

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

==> tests/test-for2.mg <==
object main {
event create
{
  int i;
  i = 0;
  for ( ; i < 5; ) {
    std::print::i(i);
    i = i + 1;
  }
  std::print::i(42);
  std::game::end();
  return;
}
}

==> tests/test-for2.out <==
0
1
2
3
4
42

==> tests/test-fordef1.mg <==
object main {
  event create
  {
    for (int i = 0 ; i < 5 ; i = i + 1)
      std::print::i(i);
    std::game::end();
    return;
  }
}

==> tests/test-fordef1.out <==
0
1
```

```
2
3
4

==> tests/test-fordef2.mg <==
object main {
event create
{
  int i = 0;
  for (int i = 1; i < 5 ; i = i + 1)
    std::print::i(i);
  std::print::i(i);
  std::game::end();
  return;
}
}


==> tests/test-fordef2.out <==
1
2
3
4
0

==> tests/test-fordef3.mg <==
object main {
event create
{
  int i = 0;
  for(; i < 5; i = i + 1)
    std::print::i(i);
  std::print::i(i);
  std::game::end();
  return;
}
}

==> tests/test-fordef3.out <==
0
1
2
3
4
5

==> tests/test-foreach2.mg <==
```

```
object aide { event create { std::print::s("aide"); } }

object main {
  bool created;
  event create {
    int i;
    create aide;
    i = 1;
    foreach(aide h)
      if (i < 5) { create aide; i = i + 1; }
    /* should create 5 aides, since new one should also be iterated over */
    std::game::end();
  }
}


==> tests/test-foreach2.out <==
aide
aide
aide
aide
aide

==> tests/test-foreach3.mg <==

object parent { }
object child : parent { }

object main {
  event create {
    for (int i = 0; i < 10; ++i)
      create parent;

    for (int i = 0; i < 10; ++i)
      create child;

    int i = 0;
    foreach (parent p) ++i; // 20 parents total
    std::print::i(i);

    i = 0;
    foreach (child c) ++i; // 10 parents total
    std::print::i(i);

    std::game::end();
  }
}
```

```
==> tests/test-foreach3.out <==
20
10

==> tests/test-foreach4.mg <==

object helper { }
object aide { }

void print_obj_count() {
  int i = 0;
  foreach (object o) ++i;
  std::print::i(i);
}

object main {
  event create {
    create helper;
    create aide;
    create aide;

    print_obj_count(); // 4 including myself
    foreach (object o) destroy o;

    print_obj_count();

    std::game::end();
  }
}


==> tests/test-foreach4.out <==
4
0

==> tests/test-foreach5.mg <==
object aide { }

object main {
  event create {
    aide a = create aide;
    create aide;

    std::print::b(std::game::obj_alive(a));
    destroy(a);
    std::print::b(std::game::obj_alive(a));
```

```
    std::game::end();
  }
}


==> tests/test-foreach5.out <==
true
false

==> tests/test-foreach.mg <==
int x;
object helper {
  int y;

  event create { x = x + 2; this.y = x; }
}

object main {
  int y;
  event create {
    this.y = 3; /* this is not std::printed since it's not a helper */

    int x;
    for (x = 0; x < 5; x = x + 1)
      create helper;

    foreach (helper h) {
      std::print::i(h.y);
    }

    std::game::end();
  }
}

==> tests/test-foreach.out <==
2
4
6
8
10

==> tests/test-func1.mg <==
int add(int a, int b)
{
  return a + b;
}

object main {
```

```
event create
{
  int a;
  a = add(39, 3);
  std::print::i(a);
  std::game::end();
  return;
}
}

==> tests/test-func1.out <==
42

==> tests/test-func2.mg <==
/* Bug noticed by Pin-Chin Huang */

int fun(int x, int y)
{
  return 0;
}

object main {
event create
{
  int i;
  i = 1;

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

  std::print::i(i);
  std::game::end();
  return;
}
}


==> tests/test-func2.out <==
3

==> tests/test-func3.mg <==
void printem(int a, int b, int c, int d)
{
  std::print::i(a);
  std::print::i(b);
  std::print::i(c);
  std::print::i(d);
}
```

```
object main {
event create
{
  printem(42,17,192,8);
  std::game::end();
  return;
}
}

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

==> tests/test-func4.mg <==
int add(int a, int b)
{
  int c;
  c = a + b;
  return c;
}

object main {
event create
{
  int d;
  d = add(52, 10);
  std::print::i(d);
  std::game::end();
  return;
}
}

==> tests/test-func4.out <==
62

==> tests/test-func5.mg <==
int foo(int a)
{
  return a;
}

object main {
event create
{
  std::game::end();
  return;
```

```
}
}
```

==> tests/test-func5.out <==

==> tests/test-func6.mg <==

```
void foo() {}

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

object main {
event create
{
  std::print::i(bar(17, false, 25));
  std::game::end();
  return;
}
}
```

==> tests/test-func6.out <==

```
42
```

==> tests/test-func7.mg <==

```
int a;

void foo(int c)
{
  a = c + 42;
}

object main {
event create
{
  foo(73);
  std::print::i(a);
  std::game::end();
  return;
}
}
```

==> tests/test-func7.out <==

```
115
```

==> tests/test-func8.mg <==

```
void foo(int a)
{
  std::print::i(a + 3);
}
```

```
object main {
event create
{
  foo(40);
  std::game::end();
  return;
}
}
```

==> tests/test-func8.out <==
```
43
```

==> tests/test-gcd2.mg <==
```
int gcd(int a, int b) {
  while (a != b)
    if (a > b) a = a - b;
    else b = b - a;
  return a;
}

object main {
event create
{
  std::print::i(gcd(14,21));
  std::print::i(gcd(8,36));
  std::print::i(gcd(99,121));
  std::game::end();
  return;
}
}
```

==> tests/test-gcd2.out <==
```
7
4
11
```

==> tests/test-gcd.mg <==
```
int gcd(int a, int b) {
  while (a != b) {
    if (a > b) a = a - b;
    else b = b - a;
  }
  return a;
}

object main {
event create
```

```
{
  std::print::i(gcd(2,14));
  std::print::i(gcd(3,15));
  std::print::i(gcd(99,121));
  std::game::end();
  return;
}
}

==> tests/test-gcd.out <==
2
3
11

==> tests/test-global1.mg <==
int a;
int b;

void printA()
{
  std::print::i(a);
}

void printB()
{
  std::print::i(b);
}

void incab()
{
  a = a + 1;
  b = b + 1;
}

object main {
event create
{
  a = 42;
  b = 21;
  printA();
  printB();
  incab();
  printA();
  printB();
  std::game::end();
  return;
}
}
```

```
==> tests/test-global1.out <==
42
21
43
22

==> tests/test-global2.mg <==
bool i;

object main {
event create
{
  int i; /* Should hide the global i */

  i = 42;
  std::print::i(i + i);
  std::game::end();
  return;
}
}

==> tests/test-global2.out <==
84

==> tests/test-global3.mg <==
int i;
bool b;
int j;

object main {
event create
{
  i = 42;
  j = 10;
  std::print::i(i + j);
  std::game::end();
  return;
}
}

==> tests/test-global3.out <==
52

==> tests/test-hello.mg <==
object main {
event create
{
```

```
  std::print::i(42);
  std::print::i(71);
  std::print::i(1);
  std::game::end();
  return;
}
}

==> tests/test-hello.out <==
42
71
1

==> tests/test-if1.mg <==
object main {
event create
{
  if (true) std::print::i(42);
  std::print::i(17);
  std::game::end();
  return;
}
}

==> tests/test-if1.out <==
42
17

==> tests/test-if2.mg <==
object main {
event create
{
  if (true) std::print::i(42); else std::print::i(8);
  std::print::i(17);
  std::game::end();
  return;
}
}

==> tests/test-if2.out <==
42
17

==> tests/test-if3.mg <==
object main {
event create
{
  if (false) std::print::i(42);
```

```
  std::print::i(17);
  std::game::end();
  return;
}
}

==> tests/test-if3.out <==
17

==> tests/test-if4.mg <==

object main {
event create
{
  if (false) std::print::i(42); else std::print::i(8);
  std::print::i(17);
  std::game::end();
  return;
}
}

==> tests/test-if4.out <==
8
17

==> tests/test-if5.mg <==
int cond(bool b)
{
  int x;
  if (b)
    x = 42;
  else
    x = 17;
  return x;
}

object main {
event create
{
 std::print::i(cond(true));
 std::print::i(cond(false));
 std::game::end();
 return;
}
}

==> tests/test-if5.out <==
42
```

```
17

==> tests/test-inc-dec1.mg <==
object main {
  event create {
    int a;
    a = 42;
    std::print::i(++a);
    ++a = 3;
    std::print::i(a);
    std::print::i(--a);
    std::print::i(--a + 5);
    std::game::end();
  }
}

==> tests/test-inc-dec1.out <==
43
3
2
6

==> tests/test-inheritance2.mg <==
object parent {
  int x;
  void compute() { std::print::i(x); }
}

object child : parent {
  event create { x = 3; compute(); } // overwrites parent x
}

object child2 : parent {
  void compute() { std::print::i(10); }
  event create { compute(); } // calls own compute
}

object main {
  event create {
    child c = create child;
    c.x = 5;
    c.compute();
    create child2;
    std::game::end();
  }
}

==> tests/test-inheritance2.out <==
```

```
3
5
10

==> tests/test-inheritance3.mg <==
object parent {
  event step { std::print::s("hello!"); }
}

object child : parent {
  event step { std::print::s("hey!"); }
}

object main {
  int i;
  event create {
    i = 2;
    create parent;
    create child;
  }

  event step {
    i -= 1; // run for 2 steps
    if (i == 0)
      std::game::end();
  }
}

==> tests/test-inheritance3.out <==
hello!
hey!
hello!
hey!

==> tests/test-inheritance4.mg <==
object parent {
  void compute() { std::print::s("parent compute"); }
  event create { std::print::s("parent create"); }
}

object child : parent {
  void compute() { std::print::s("child compute"); }
  event create { std::print::s("child create"); }
}

object main {
  int i;
  event create {
```

```
    child c = create child; // child create
    c.compute();            // child compute

    parent p = create parent; // parent create
    p.compute();              // parent compute

    p = c;
    p.compute(); // parent compute, despite pointing to child, since non-
virtual
    std::game::end();
  }
}

==> tests/test-inheritance4.out <==
child create
child compute
parent create
parent compute
parent compute

==> tests/test-inheritance5.mg <==
object parent {
  event create  { std::print::s("create"); }
  event step    { std::print::s("step"); }
  event draw    { std::print::s("draw"); }
  event destroy { std::print::s("destroy"); }
}

object child : parent { } // nothing should be overridden

object main {
  int i;
  child c;
  event create {
    i = 0;
    c = create child;
  }

  event step {
    if (i >= 1) { destroy c; std::game::end(); }
    else ++i;
  }
}

==> tests/test-inheritance5.out <==
create
step
draw
```

```
destroy

==> tests/test-inheritance6.mg <==
object parent {
  event create(int x)  { std::print::i(x); }
}

object daughter : parent { }
object son : parent {
  event create { super(5); } // super with arguments from parent
}

object granddaughter : daughter { }
object grandson : daughter {
  event create {
    super(6); // super with arguments from grandparent
  }
}

object main {
  event create {
    create daughter(3); // create with arguments from parent
    create granddaughter(4); // create with arguments from grandparent
    create son;
    create grandson;
    std::game::end();
  }
}

==> tests/test-inheritance6.out <==
3
4
5
6

==> tests/test-inheritance7.mg <==
object B : A::C { }

namespace A {
  object C : A::D { }
  namespace A { object D { } }
}

object main {
  event create {
    A::A::D obj = create B;
    std::print::s("success!");
    std::game::end();
```

```
  }
}

==> tests/test-inheritance7.out <==
success!

==> tests/test-inheritance8.mg <==
object parent {
  event destroy { std::print::s("parent destroy"); }
}

object son : parent {
  event destroy { std::print::s("son destroy"); }
}

object daughter : parent {
  event destroy { std::print::s("daughter destroy"); super(); super(); }
}

object main {
  event create {
    std::print::s("destroy");
    create son;
    create daughter;
    foreach (parent p) destroy p;

    std::print::s("delete");
    create son;
    create daughter;
    create parent;
    foreach (parent p) delete p;

    std::game::end();
  }
}

==> tests/test-inheritance8.out <==
destroy
son destroy
daughter destroy
parent destroy
parent destroy
delete

==> tests/test-inheritance.mg <==
object main {
  event create {
    std::print::s("success");
```

```
    std::game::end();
  }
}

// a deeply nested string of objects will not raise a false positive on
// inheritance cycles
object A : S::A { }

namespace S {
  object A : S::A { }
  namespace S {
    object A : S::A { }
    namespace S {
      object A { }
    }
  }
}


==> tests/test-inheritance.out <==
success

==> tests/test-local1.mg <==
void foo(bool i)
{
  int i; /* Should hide the formal i */

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

object main {
event create
{
  foo(true);
  std::game::end();
  return;
}
}

==> tests/test-local1.out <==
84

==> tests/test-local2.mg <==
int foo(int a, bool b)
{
  int c;
  bool d;
```

```
  c = a;

  return c + 10;
}

object main {
event create {
 std::print::i(foo(37, false));
 std::game::end();
 return;
}
}

==> tests/test-local2.out <==
47

==> tests/test-local-scope.mg <==
bool a;
object main {
  int a;
  event create {
    a = 5;
    std::print::i(a);
    std::game::end();
  }
}

==> tests/test-local-scope.out <==
5

==> tests/test-minusasn1.mg <==

int minusAsn(int x)
{
  return x -= 2;
}

object main {
event create
{
  std::print::i( minusAsn(2) );

  std::game::end();
  return;
}
}
```

```
==> tests/test-minusasn1.out <==
0

==> tests/test-minusasn2.mg <==

object main {
event create
{
  float x;
  x = 3.0;
  x -= 3.0;
  std::print::f( x );

  std::game::end();
}
}

==> tests/test-minusasn2.out <==
0.000000

==> tests/test-minusasn3.mg <==

int minusAsn(int x)
{
  int y;
  y = ((x -= 2) = 3);
  return y;

}

object main {
event create
{
  std::print::i( minusAsn(1) );

  std::game::end();
  return;
}
}

==> tests/test-minusasn3.out <==
3

==> tests/test-minusasn4.mg <==

int minusAsn(int x)
{
  return (x -= 2) -= 2;
```

```
}

object main {
event create
{
  std::print::i( minusAsn(0) );

  std::game::end();
  return;
}
}

==> tests/test-minusasn4.out <==
-4

==> tests/test-modulo.mg <==
using std;
namespace print = std::print;

object main {
  event create {
    print::i(5 % 2);
    print::i(-5 % 2);
    print::f(6 % 2.5);
    print::f(5.5 % 2);
    print::f(5.5 % 2.5);
    std::game::end();
  }
}

==> tests/test-modulo.out <==
1
-1
1.000000
1.500000
0.500000

==> tests/test-namespace2.mg <==
namespace A {
  int y; int z;
  void compute(int x) { std::print::s("A"); std::print::i(x);
std::print::i(y); }
  void compute2(int x) { std::print::s("A2"); std::print::i(x);
std::print::i(z); }
}

namespace B {
  void compute() { std::print::s("B"); }
```

```
}

void compute2() { std::print::s("main2"); }

using A;
using B;
int z;

object main {
  event create {
    compute(); // B
    y = 5; // overwrites A::z
    A::compute(10); // A 10 5
    compute2(); // main2
    A::z = 5;
    z = 10; // does not overwrite A::z
    A::compute2(20); // A2 20 5
    std::game::end();
  }
}

==> tests/test-namespace2.out <==
B
A
10
5
main2
A2
20
5

==> tests/test-namespace3.mg <==
namespace A {
  object C { }
  object B {
    event create(C c) { std::print::s("create success"); }
    void compute(B b) { std::print::s("member success"); }
  }
  void compute(B b) { std::print::s("success"); }
}

object main {
  event create {
    A::B b = create A::B(create A::C);
    b.compute(b);
    A::compute(b);

    std::game::end();
```

```
  }
}

==> tests/test-namespace3.out <==
create success
member success
success

==> tests/test-namespace.mg <==

int compute () { return 1; }

namespace a {
  int compute() { return 2; }
  aide b;
  object aide {
    int x;
    event create { x = 4; b = this; }
    int compute() { return 3; }
    aide make() { return create aide; }
    a::aide inner() { return create a::aide; }
  }

  namespace a {
    object aide {
      int y; int x;
      int compute() { return 6; }
      event create { x = 5; }
    }
  }
}

object main {
  event create {
    // no namespace specified gives regular fn
    std::print::i(compute());
    // appropriate outer namespace call
    std::print::i(a::compute());
    // returning an object defined in the namespace of the call
    std::print::i(a::b.make().compute());
    // member call from namespace
    std::print::i(a::b.compute());
    // member variable from namespace
    std::print::i(a::b.make().x);
    // returning an object defined inside a ns inside the ns of the call
    std::print::i(a::b.inner().x);
    std::print::i(a::b.inner().compute());
    std::game::end();
```

```
  }
}

==> tests/test-namespace.out <==
1
2
3
3
4
5
6

==> tests/test-ns-alias2.mg <==

namespace A = a; // alias can be defined before what it refers to.
namespace a { int x; }

object main {
  event create {
    a::x = 1;
    std::print::i(A::x);
    std::game::end();
  }
}

==> tests/test-ns-alias2.out <==
1

==> tests/test-ns-alias.mg <==

namespace a {
  namespace b { int x; }
  namespace c {
    int x;
    namespace d { int x; }
  }
  namespace D = c::d; // inner alias
}

namespace B = a::b; // regular alias
namespace BB = B; // alias to alias
namespace C = a::c;
namespace D = a::c::d; // alias to nested
namespace DD = a::D; // alias to alias of inner ns

object main {
  event create {
    a::b::x = 1;
```

```
    std::print::i(B::x);
    std::print::i(BB::x);
    a::c::x = 2;
    std::print::i(C::x);
    a::c::d::x = 3;
    std::print::i(a::D::x);
    std::print::i(D::x);
    std::print::i(DD::x);
    std::game::end();
  }
}

==> tests/test-ns-alias.out <==
1
1
2
3
3
3

==> tests/test-objfn1.mg <==

int number() { return 3; }

object helper {
  int number() { return 4; }
  int get_number() { return number(); } // number is shadowed in class helper
  event create { }
}
object main {
  helper make() { return create helper; }
  event create {
    helper h;
    h = make(); // self member function test
    h = this.make();

    std::print::i(h.number()); // other member function test
    std::print::i((create helper).number());

    std::print::i(h.get_number());
    std::print::i(number()); // number is not shadowed in object main

    std::game::end();
  }
}

==> tests/test-objfn1.out <==
4
```

```
4
4
3

==> tests/test-objfn2.mg <==
int x;
int get_global() { return x; }
int set_global(int y) { x = y; }

object main {
  int x;
  int set(int x) {
    // manipulates member x with parameter x
    this.x = x;

    // manipulates local x
    int x = x + 1;
    std::print::i(x);
  }

  event create {
    // manipulates member x
    set(2);
    std::print::i(x);

    // manipulates global x
    set_global(1);
    std::print::i(get_global());
    std::game::end();
  }
}

==> tests/test-objfn2.out <==
3
2
1

==> tests/test-ops1.mg <==
object main {
event create
{
  std::print::i(1 + 2);
  std::print::i(1 - 2);
  std::print::i(1 * 2);
  std::print::i(100 / 2);
  std::print::i(99);
  std::print::b(1 == 2);
  std::print::b(1 == 1);
```

```
  std::print::i(99);
  std::print::b(1 != 2);
  std::print::b(1 != 1);
  std::print::i(99);
  std::print::b(1 < 2);
  std::print::b(2 < 1);
  std::print::i(99);
  std::print::b(1 <= 2);
  std::print::b(1 <= 1);
  std::print::b(2 <= 1);
  std::print::i(99);
  std::print::b(1 > 2);
  std::print::b(2 > 1);
  std::print::i(99);
  std::print::b(1 >= 2);
  std::print::b(1 >= 1);
  std::print::b(2 >= 1);
  std::game::end();
}
}

==> tests/test-ops1.out <==
3
-1
2
50
99
false
true
99
true
false
99
true
false
99
true
true
false
99
false
true
99
false
true
true

==> tests/test-ops2.mg <==
```

```
object main {
event create
{
  std::print::b(true);
  std::print::b(false);
  std::print::b(true && true);
  std::print::b(true && false);
  std::print::b(false && true);
  std::print::b(false && false);
  std::print::b(true || true);
  std::print::b(true || false);
  std::print::b(false || true);
  std::print::b(false || false);
  std::print::b(!false);
  std::print::b(!true);
  std::print::i(-10);
  std::print::i(-(-42));
  std::game::end();
}
}

==> tests/test-ops2.out <==
true
false
true
false
false
false
true
true
true
false
true
false
-10
42

==> tests/test-retcb2.mg <==

int compute() {
  int i;

  i = 15;
  std::print::i(i);
  return i;

  i = 32; /* code after a return is ok */
  std::print::i(i);
```

```
}

object main {
event create
{
  compute();
  std::game::end();
}
}

==> tests/test-retcb2.out <==
15

==> tests/test-retcb.mg <==

int compute() {
  int i;
  {
    i = 15;
    std::print::i(i);
    return i; /* ok: return at end of block */
  }
  i = 3;
  std::print::i(i);
  return i;
}
object main {
event create
{
  compute();
  std::game::end();
}
}

==> tests/test-retcb.out <==
15

==> tests/test-scope.mg <==

object main {
event create {
  int x;
  x = 3;
  std::print::i(x);

  int y;
  y = 4;
  std::print::i(y);
```

```
  if (x == 3) {
    std::print::i(x);
    int x; // OK: can redeclare in different block
    x = 5;
    std::print::i(x);
  }
  std::print::i(x);
  std::game::end();
}
}

==> tests/test-scope.out <==
3
4
3
5
3

==> tests/test-sifp1.mg <==
int add(int a, int b)
{
  return a + b;
}

object main {
event create
{
  std::print::f(3+3.5);
  std::print::f(3.5+3);
  std::print::f(3+3);
  int b = 4.5;
  float c = b;
  std::print::f(c);
  std::print::i(add(39, 3.5));
  if (3.0 == 3) {
    std::print::s("3.0 == 3\n");
  }
  else {
    std::print::s("3.0 != 3\n");
  }
  std::game::end();
}
}

==> tests/test-sifp1.out <==
6.500000
6.500000
```

```
6.000000
4.000000
42
3.0 == 3


==> tests/test-string.mg <==
object main {
event create {
  string c;
  c = "hello world";
  std::print::s(c);
  std::print::s("goodbye world");
  std::print::s("this is \"right\"");
  std::print::s("Either\\or is okay");
  std::print::s("multiple\nlines");
  std::game::end();
  return;
}
}

==> tests/test-string.out <==
hello world
goodbye world
this is "right"
Either\or is okay
multiple
lines

==> tests/test-super1.mg <==
object son : parent {
  event step {
    std::print::s("child step");
    super();
  }

  event draw {
    std::print::s("child draw");
    super();
  }
}

object daughter : parent {
  event step {
    super();
    std::print::s("daughter step");
  }
```

```
  event draw {
    super();
    std::print::s("daughter draw");
  }
}

object parent {
  event step { std::print::s("parent step"); }
  event draw { std::print::s("parent draw"); }
}

object main {
  int i;
  event create {
    i = 0;
    create son;
    create daughter;
  }

  event step { std::game::end(); }
}

==> tests/test-super1.out <==
child step
parent step
parent step
daughter step
child draw
parent draw
parent draw
daughter draw

==> tests/test-this2.mg <==

void set(main m) {
  m.x = 3;
}

object main {
  int x;
  event create {
    set(this);
    std::print::i(this.x);
    std::game::end();
  }
}

==> tests/test-this2.out <==
```

```
3

==> tests/test-this.mg <==

object main {
  int x;
  event create { this.x = 3; }
  event step {
    int x;
    x = 5;
    std::print::i(this.x);
    std::print::i(x);
    std::game::end();
  }
}

==> tests/test-this.out <==
3
5

==> tests/test-this-scope.mg <==

object main {
  int x;
  int y;
  event create {
    x = 0;
    this.y = 1;
    std::print::i(this.x);
    std::print::i(y);

    int y;
    y = 2;
    std::print::i(y);
    std::print::i(this.y);

    this.y = 3;
    std::print::i(y);
    std::print::i(this.y);

    std::game::end();
  }
}

==> tests/test-this-scope.out <==
0
1
2
```

```
1
2
3

==> tests/test-var1.mg <==
object main {
event create
{
  int a;
  a = 42;
  std::print::i(a);
  std::game::end();
  return;
}
}

==> tests/test-var1.out <==
42

==> tests/test-var2.mg <==
int a;

void foo(int c)
{
  a = c + 42;
}

object main {
event create
{
  foo(73);
  std::print::i(a);
  std::game::end();
  return;
}
}

==> tests/test-var2.out <==
115

==> tests/test-vdef1.mg <==
object main {
  event create
  {
    int a = 1 + 1;
    std::print::i( a );

    std::game::end();
```

```
  }
}

==> tests/test-vdef1.out <==
2

==> tests/test-while1.mg <==
object main {
event create
{
  int i;
  i = 5;
  while (i > 0) {
    std::print::i(i);
    i = i - 1;
  }
  std::print::i(42);
  std::game::end();
  return;
}
}

==> tests/test-while1.out <==
5
4
3
2
1
42

==> tests/test-while2.mg <==
int foo(int a)
{
  int j;
  j = 0;
  while (a > 0) {
    j = j + 2;
    a = a - 1;
  }
  return j;
}

object main {
event create
{
  std::print::i(foo(7));
  std::game::end();
  return;
```

```
}
}

==> tests/test-while2.out <==
14

==> test-helpers/bad-parse.mg <==
this is not a makergame file

==> test-helpers/bad-semant.mg <==
int compute() { return "hello"; }

==> test-helpers/circular-inherit1.mg <==
namespace S = open "circular-inherit2.mg";
object A : S::B { }

==> test-helpers/circular-inherit2.mg <==
namespace S = open "circular-inherit1.mg";
object B : S::A { }

==> test-helpers/circular.mg <==
namespace c = open "circular.mg";

int x;

==> test-helpers/private.mg <==
private namespace a {
  int x;
}

==> test-helpers/recursive.mg <==
namespace this_file = open "recursive.mg";
namespace rec = this_file::rec;

==> test-helpers/simple2.mg <==
namespace one = open "simple.mg";

int x;

==> test-helpers/simple.mg <==
int x;
```