GOBLIN

Turn-based adventure games

Kevin Xiao  Manager
Bayard Neville  Language Guru
Christina Floristean  System Architect
Gabriel Uribe  Tester
Problem

- General-purpose languages have steep learning curves and are not focused on game development
- Game engines like Unity require beginners to learn both the environment and new languages
- Not friendly for new coders
What is Goblin?

- Language for simple turn-based games without extensive knowledge of software development
- Follows an abridged object oriented model
- Runs with an underlying game loop
Program Structure

- Gamers think of adventure games in terms of entities in a world that perform functions
- Adapted this model for our program structure

```
world[x,y]{
    ...
}
entities{
    ...
}
functions{
    ...
}
```
Entities

- Classes that represent game characters
- Build block is a constructor
- Does block is a method called every turn of game loop
- Special Player entity that user controls

```plaintext
entities{
  <character>:player{
    <fields>
    build{
      <variable declarations>
      <statements>
    }
    does{
      <variable declarations>
      <statements>
    }
  }
}
```
World

- Function that defines and sets up game board
- Instantiates entities by placing at coordinates on the board

```plaintext
world[x,y]{
    <variable declarations>
    <statements>
}
```
Built-in Functions

- `place()`: instantiate entity on game board
- `peek()`: returns entity pointer at coordinate
- `move()`: moves entity to a different coordinate
- `remove()`: frees entity
- `getKey()`: returns user input from terminal, written in C
- `exit`: keyword for quitting on win

```c
place(String e, num r, num c);
peek(num r, num c);
move(Entity e, num r, num c);
remove(Entity e);
row(Entity e);
col(Entity e);
getKey();
exit;
```
Abstract Syntax Tree

- program
  - world
  - entities
  - functions
Game Loop

- Abstracted from the Goblin programmer
- `main()` function that is appended to functions in the AST
- Iterates through World and calls the “does” method for every entity
- Renders World in terminal
Translator Architecture

game.gob → scanner.ml → parser.ml → ast.ml

LLVM Bytecode

getKey.c & clearScreen.c

semant.ml

sast

codegen.ml

Executable

Linking

LLVM Bytecode
Testing

- Learned that test driven development is important
- Initial complications with testing due to insertion of game loop
- Fixed towards the end
Future

- Inheritance for entities
- Multiple worlds
- Worlds of different shapes
Lessons Learned

- Create a MVP first
- Then iterate agilely on version 1.0
- Be punctual
- And of course, start early