Team 19: Skit

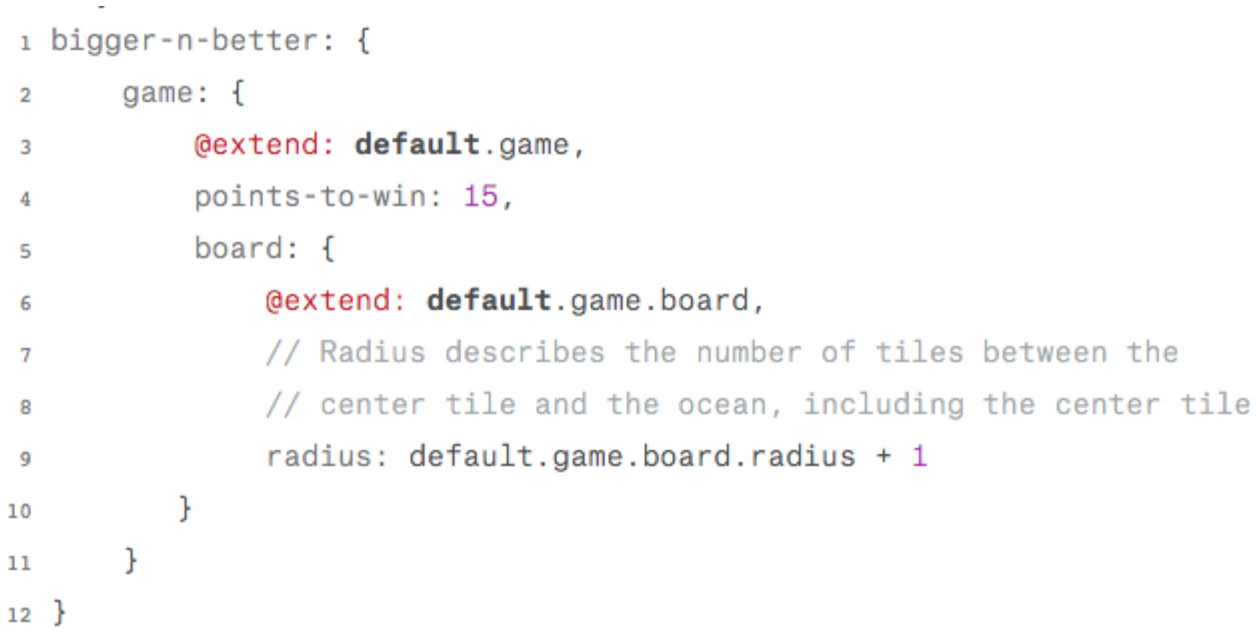
Andrew Figpope: Project Manager
Michelle Zhang: Language Guru
Márcio Paiva: System Architect
Thomas Huzij: System Integrator

THE SETTLERS OF CATAN

Skit

The Settlers of Catan Customization Kit Language

- There exists numerous ways to set up and play Settlers, including using custom boards, new rules, expansion packs, and spinoffs
- Skit is a language that is tailored to building customized *Settlers of Catan* games
- Allows users to to tweak or redefine their behaviors in a simple, straightforward, JSON-like syntax



```
bigger-n-better: {
    game: {
        @extend: default.game,
        points-to-win: 15,
        board: {
            @extend: default.game.board,
            // Radius describes the number of tiles between the
            // center tile and the ocean, including the center tile
            radius: default.game.board.radius + 1
        }
    }
}
```

accessible

flexible

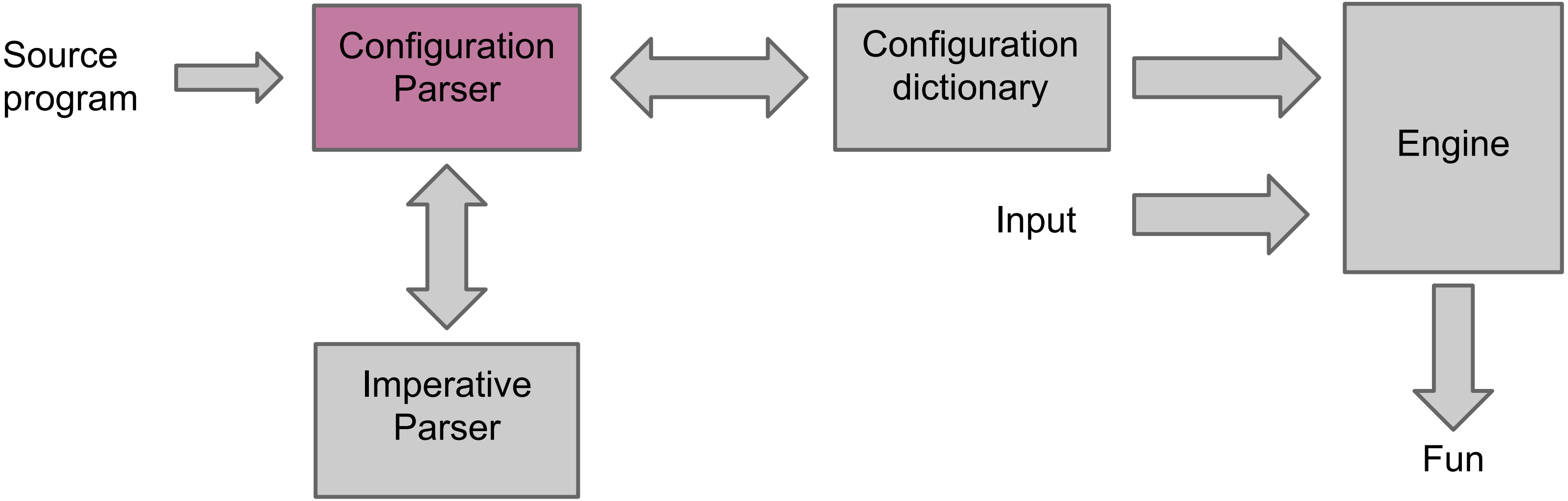
Skit

easy to read

easy to write

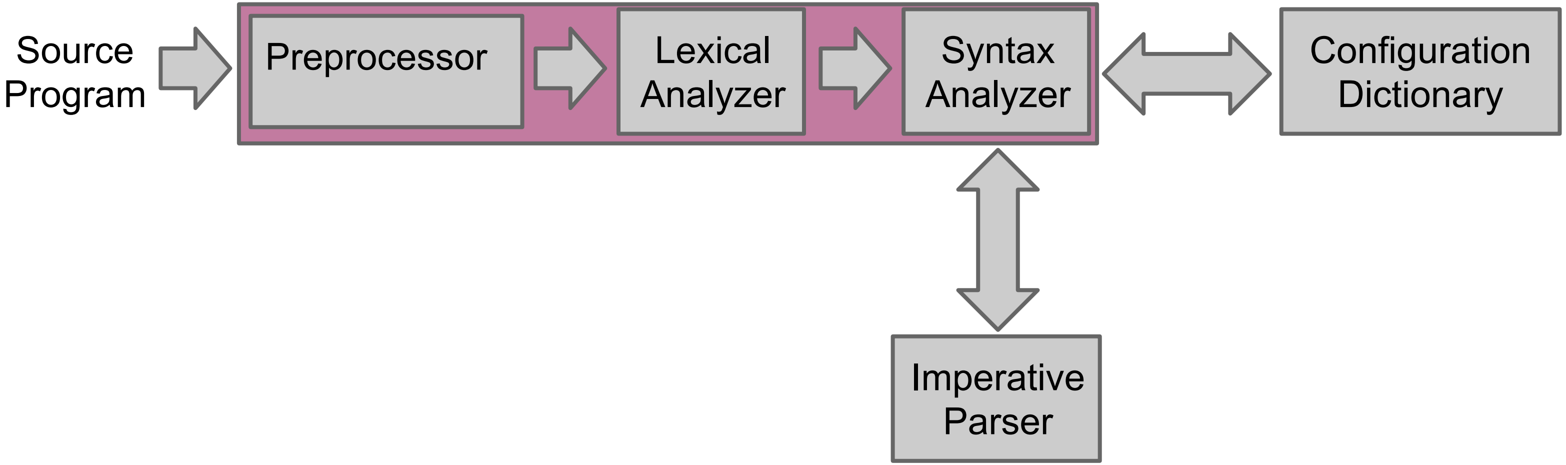
Translator Architecture

- Skit uses two intercommunicating translators to generate the configuration dictionary stored in a Python dict, which is then loaded into the engine



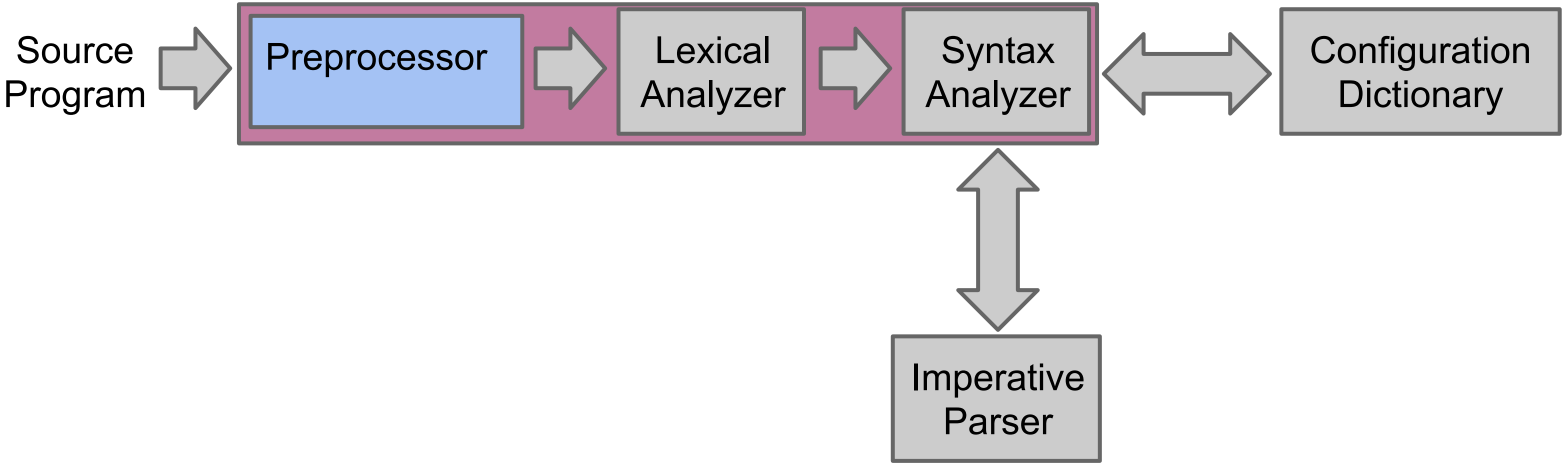
Configuration Parser

- Digging in deeper, you can see that what we refer to as the Configuration Parser obviously includes a preprocessor, a lexical analyzer, and a syntax analyzer.



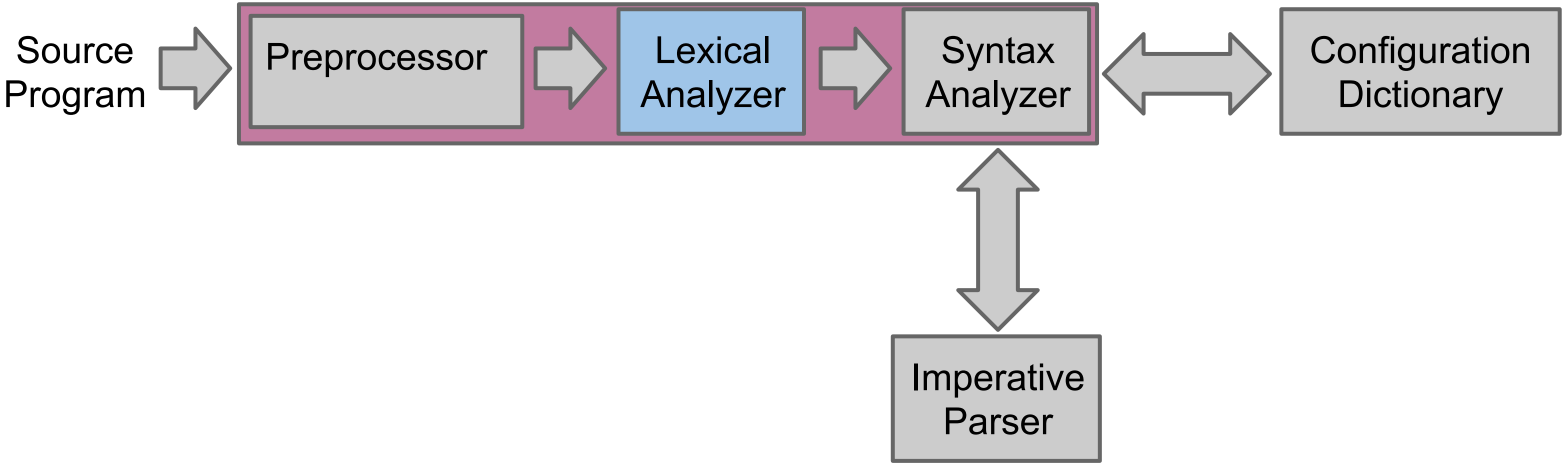
Configuration Parser

- The preprocessor is responsible for handling @import statements.



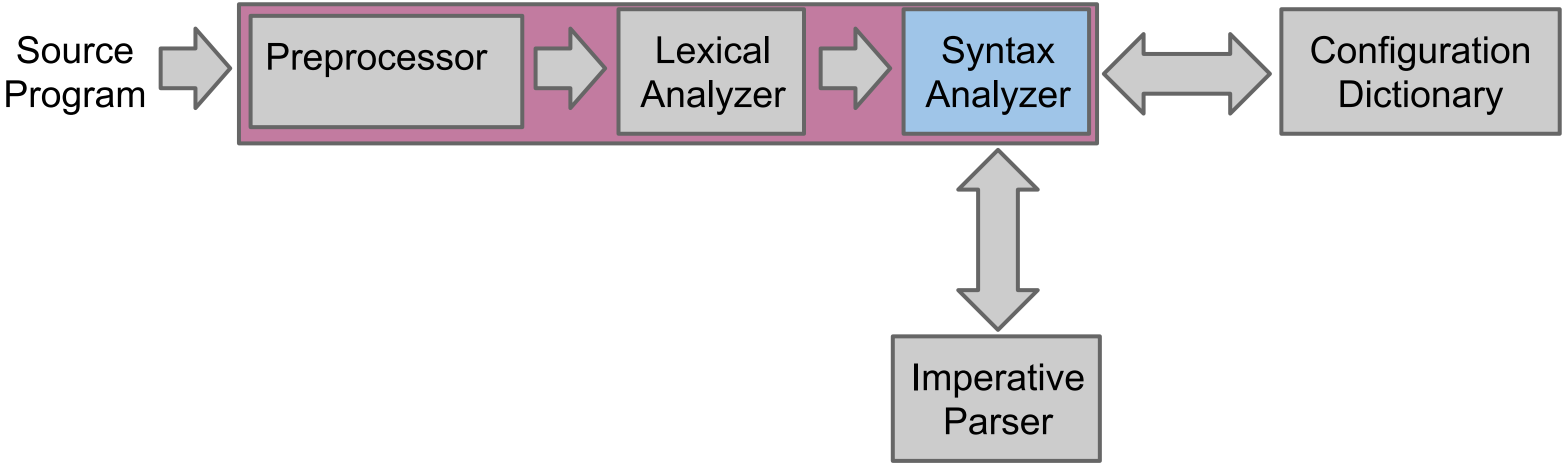
Configuration Parser

- The lexical analyzer was pretty straightforward.
- The one exception: how it tokenized the imperative function definitions.



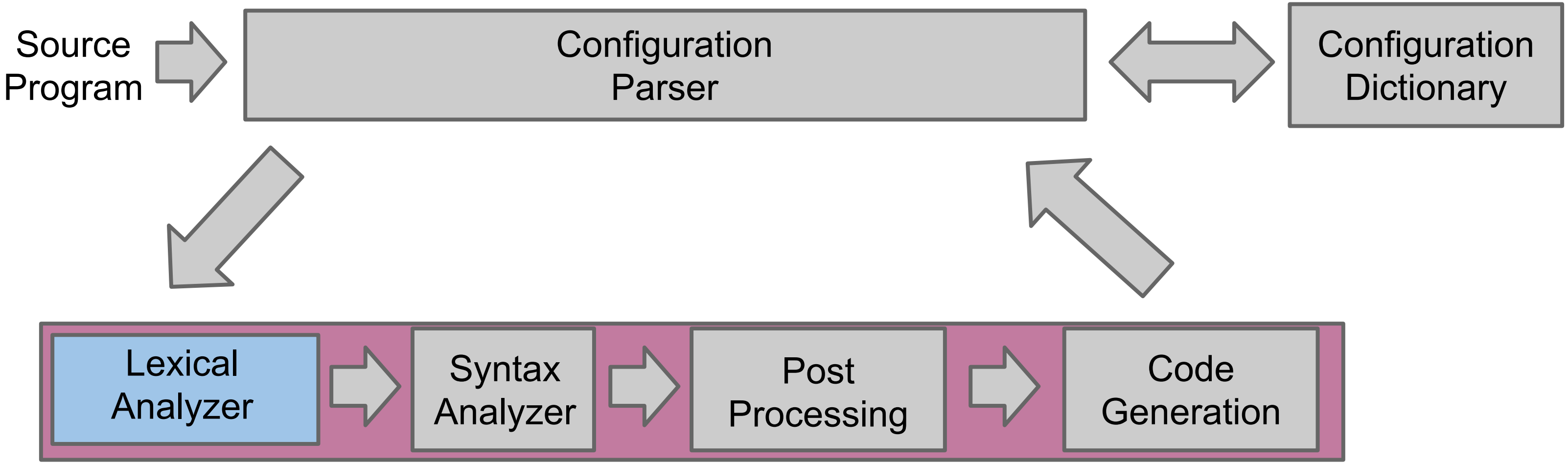
Configuration Parser

- Whenever a function token is encountered, the configuration parser just passes it to the imperative parser and expects a Python function object in return.



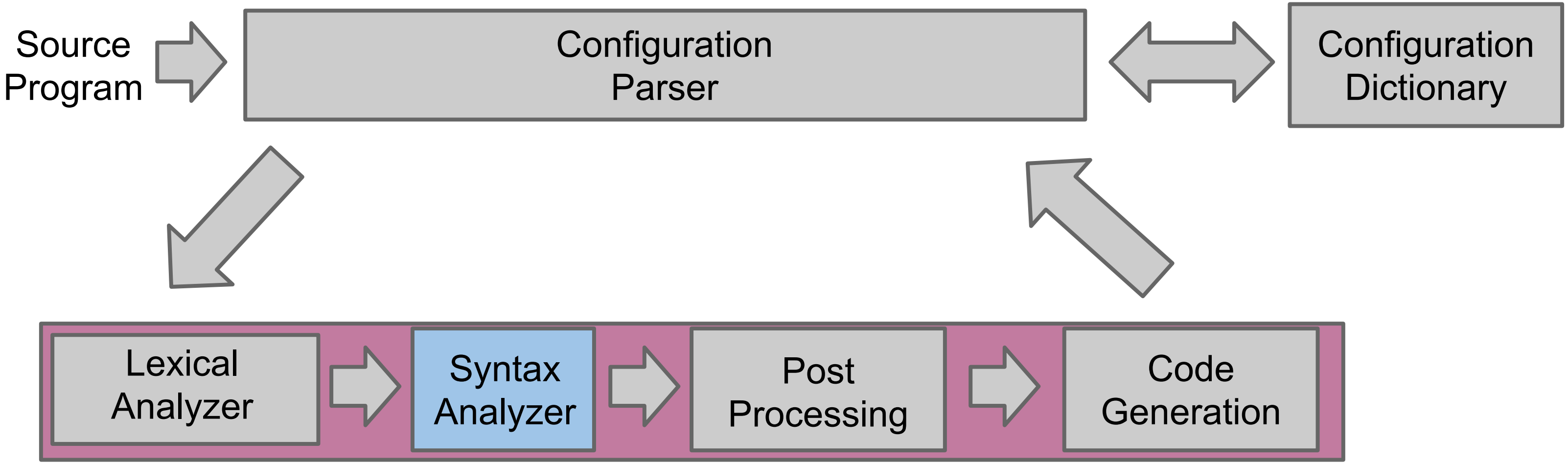
Imperative Parser

- The imperative parser is only invoked to parse a Skit function into a Python function, and tokenizes the input into the operator classes standard to most languages



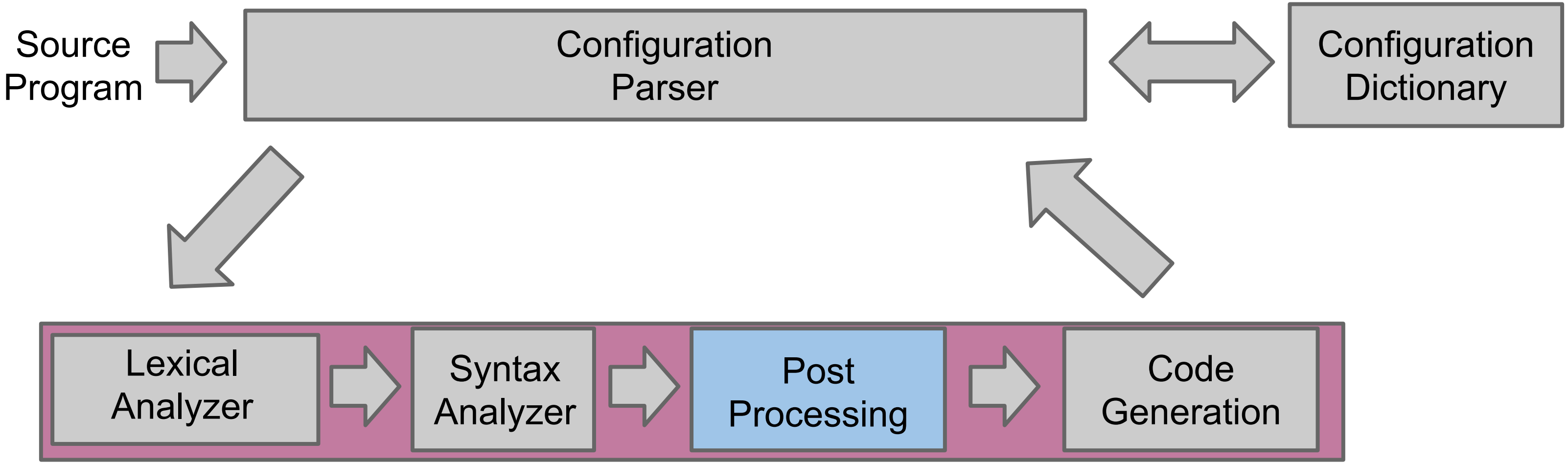
Imperative Parser

- Syntax-directed translation was then used to parse the Skit grammar directly into Python ASTs



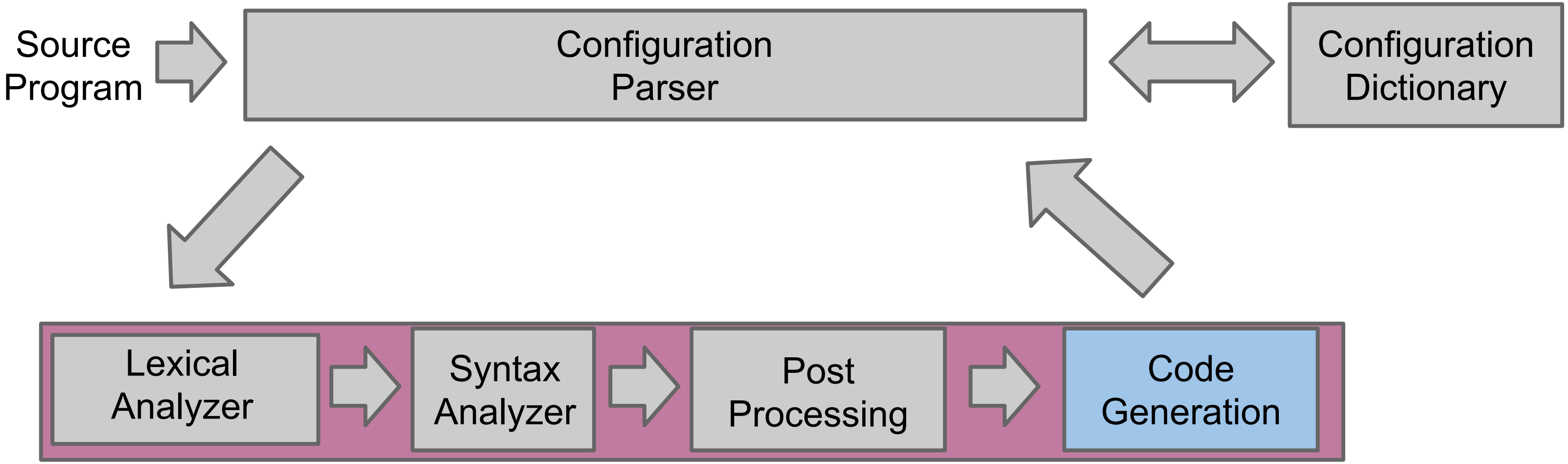
Imperative Parser

- After translation, references to parameters of the top-level function are replaced with Oracle calls to facilitate dependency injection



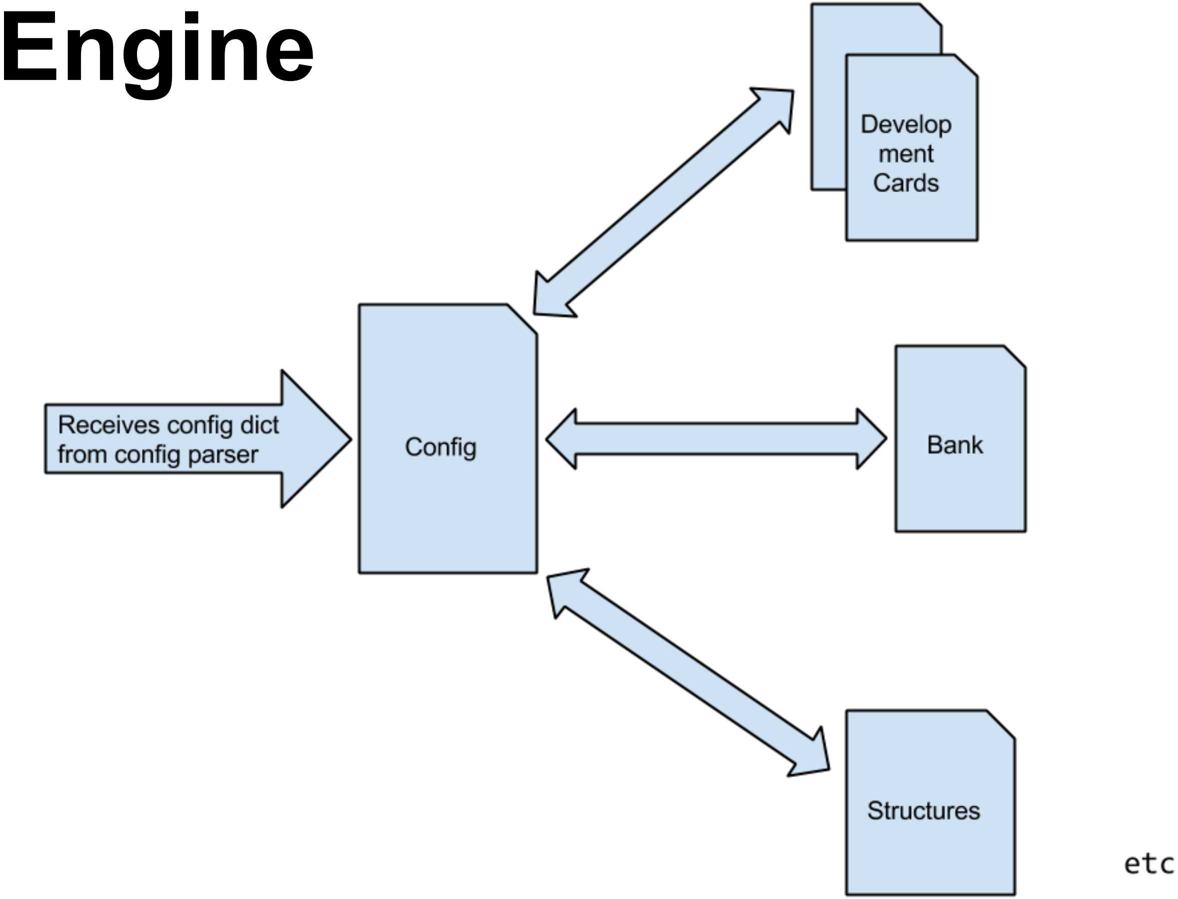
Imperative Parser

- The last stage is the execution of the AST representing the definition of the function in an environment where the Oracle is present, facilitating late-binding

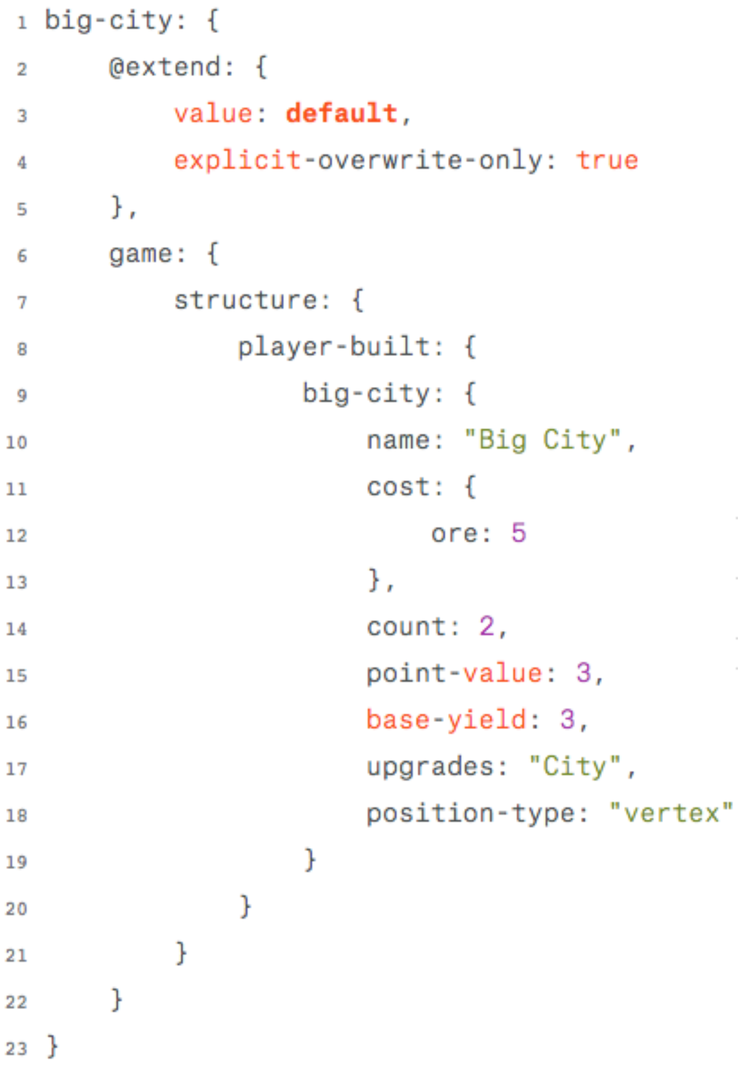


The Engine

- The dictionary parsed and translated by the configuration and imperative parsers working together is placed on a static Config class
- The config is then accessed by classes throughout the entire engine to initialize member values and instantiate different objects

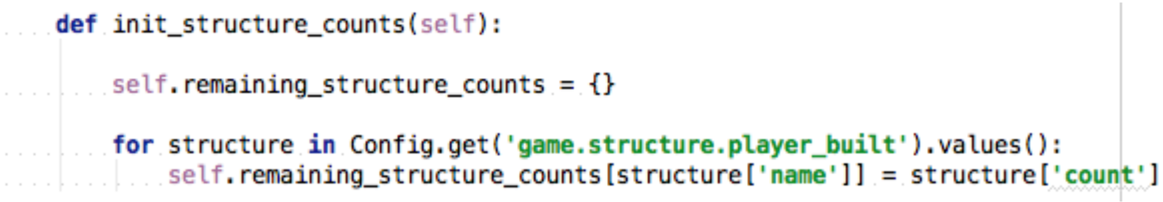


An Example



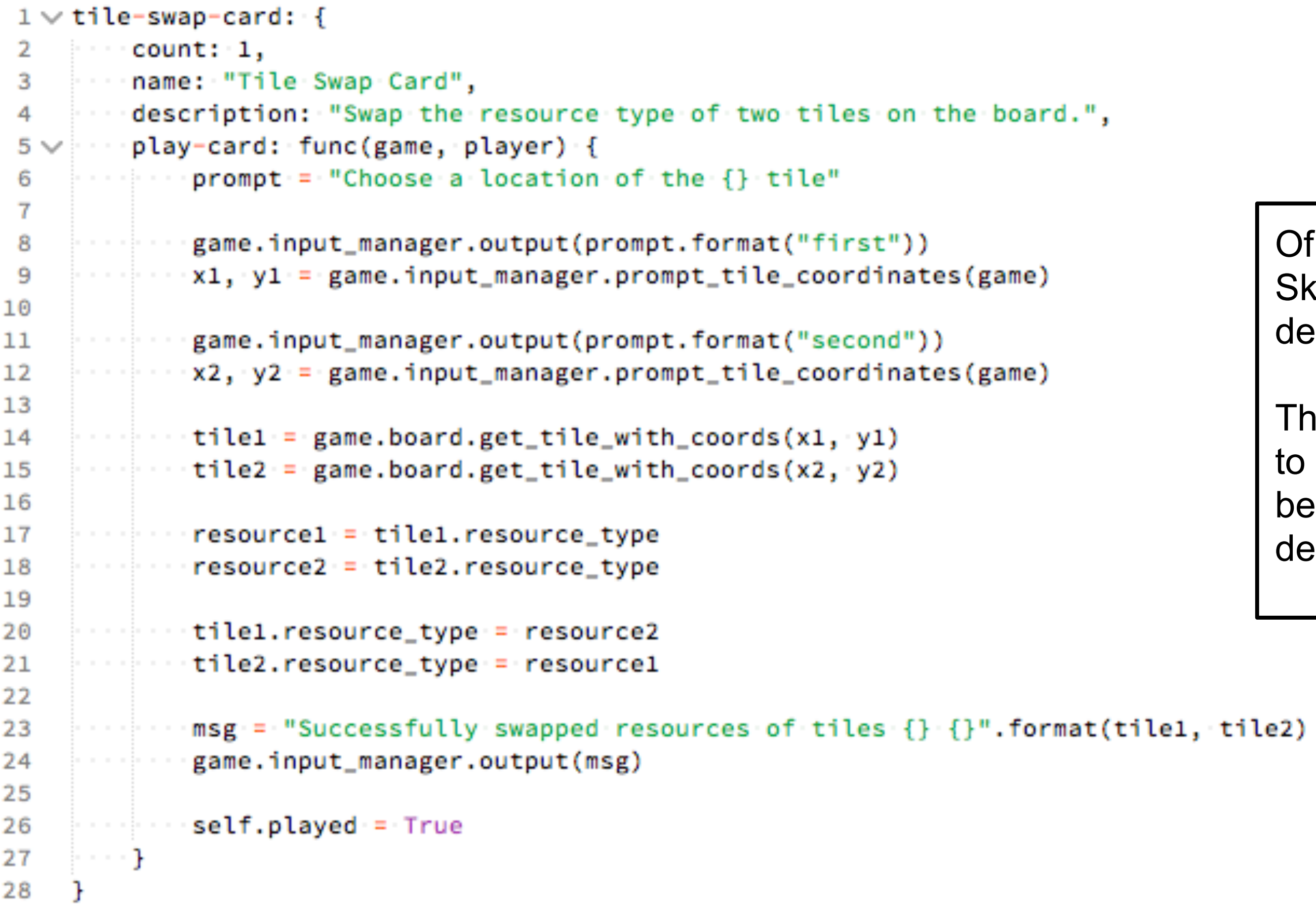
```
big-city: {
    @extend: {
        value: default,
        explicit-overwrite-only: true
    },
    game: {
        structure: {
            player-built: {
                big-city: {
                    name: "Big City",
                    cost: {
                        ore: 5
                    },
                    count: 2,
                    point-value: 3,
                    base-yield: 3,
                    upgrades: "City",
                    position-type: "vertex"
                }
            }
        }
    }
}
```

- In addition to the default player-built structures, now the Config dictionary will also have an entry for a Big City structure.
- This dictionary entry is accessed e.g. in the player class when allocating structures to players, i.e.



```python
def init_structure_counts(self):

    self.remaining_structure_counts = {}

    for structure in Config.get('game.structure.player_built').values():
        self.remaining_structure_counts[structure['name']] = structure['count']
```

Another Example



```
tile-swap-card: {
    count: 1,
    name: "Tile Swap Card",
    description: "Swap the resource type of two tiles on the board.",
    play-card: func(game, player) {
        prompt = "Choose a location of the {} tile"

        game.input_manager.output(prompt.format("first"))
        x1, y1 = game.input_manager.prompt_tile_coordinates(game)

        game.input_manager.output(prompt.format("second"))
        x2, y2 = game.input_manager.prompt_tile_coordinates(game)

        tile1 = game.board.get_tile_with_coords(x1, y1)
        tile2 = game.board.get_tile_with_coords(x2, y2)

        resource1 = tile1.resource_type
        resource2 = tile2.resource_type

        tile1.resource_type = resource2
        tile2.resource_type = resource1

        msg = "Successfully swapped resources of tiles {} {}".format(tile1, tile2)
        game.input_manager.output(msg)

        self.played = True
    }
}
```

Of course, users can also use Skit to set custom behavior by defining functions

The play-card function defined to the left, for example, would be run during a call to e.g. development_card.play_card()

Project Management

Project Management

Initially:

- Delegation of tasks was vague
- Not much accountability
- Very broad objectives
- Code disorganized
- **Ended up behind the schedule**

Project Management

Restructure:

- Very specific tasks. Deadlines
- Code style guide
- Rewrote everything from scratch
- Code reviews established
- **Productivity went up**

Development Environment

- Python 2.7.6
- PLY 3.6
- Local Mac OS X / Ubuntu

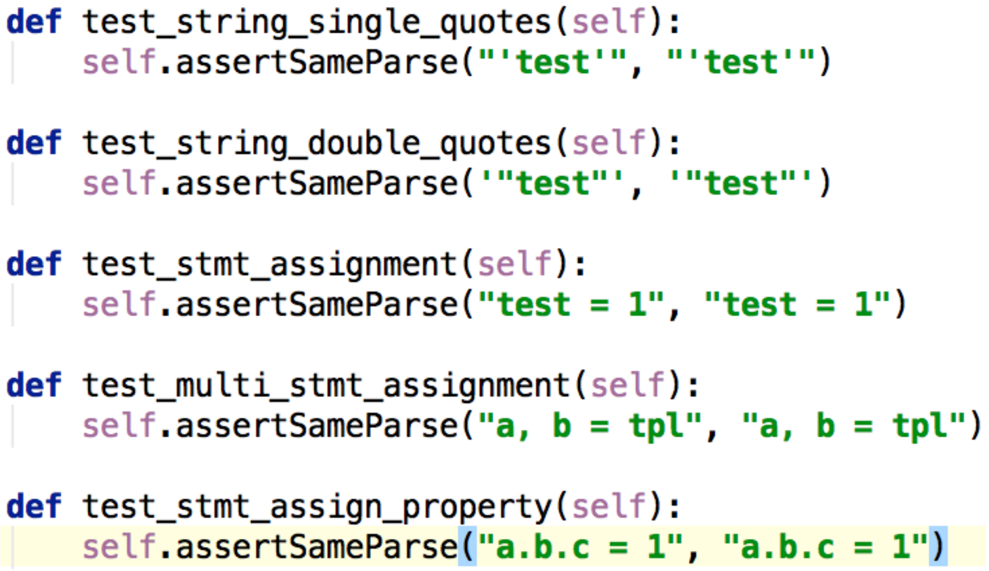
Compiler-generator tools

Began w/ the standard Lex + Yacc, but added some metaprogramming magic:

- Trivial production generation
- Registry of trivial productions
- Automatic grammar composition

Testing

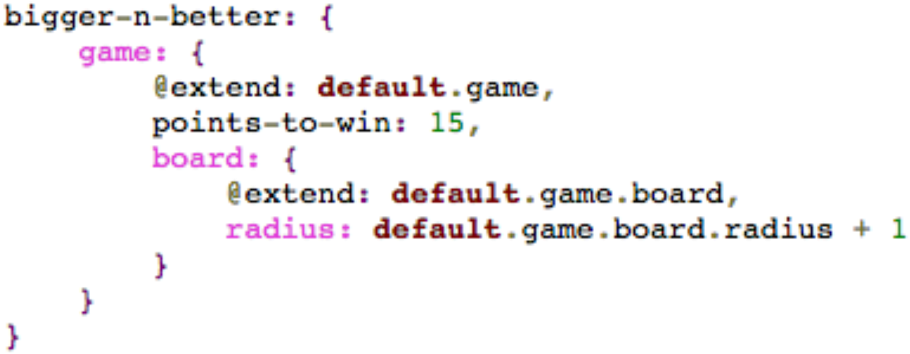
- Imperative parser compared ASTs generated by Skit to ASTs generated by Python Code



```python
def test_string_single_quotes(self):
    self.assertSameParse("'test'", "'test'")

def test_string_double_quotes(self):
    self.assertSameParse('"test"', '"test"')

def test_stmt_assignment(self):
    self.assertSameParse("test = 1", "test = 1")

def test_multi_stmt_assignment(self):
    self.assertSameParse("a, b = tpl", "a, b = tpl")

def test_stmt_assign_property(self):
    self.assertSameParse("a.b.c = 1", "a.b.c = 1")
```

Testing

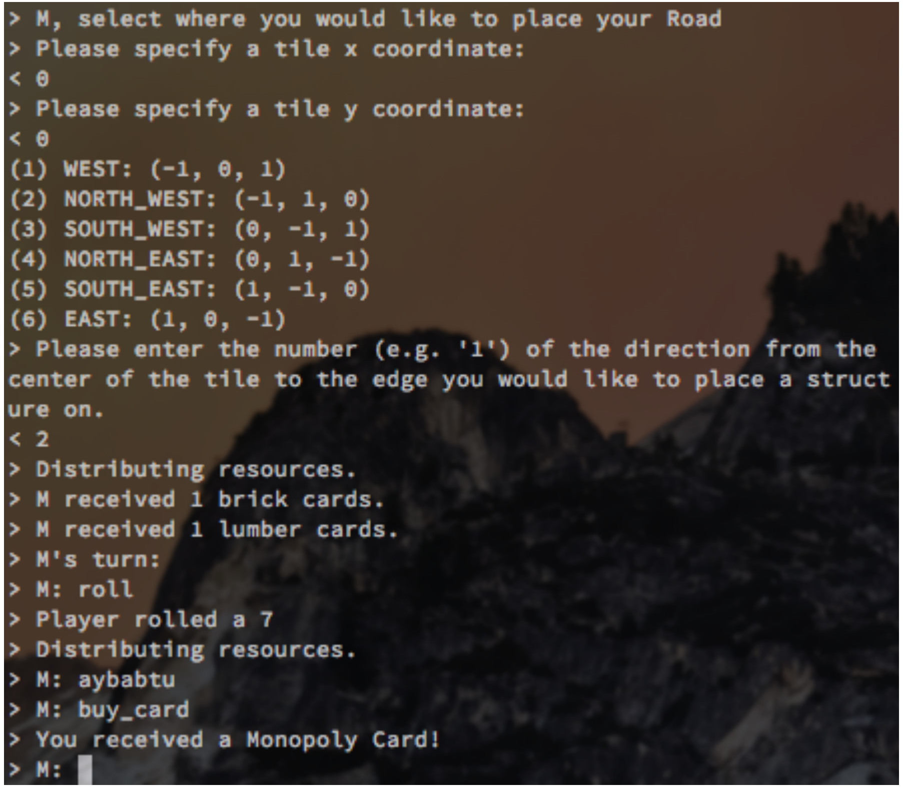
- Configuration parser was hand tested with example .skit files



```
bigger-n-better: {
    game: {
        @extend: default.game,
        points-to-win: 15,
        board: {
            @extend: default.game.board,
            radius: default.game.board.radius + 1
        }
    }
}
```

Testing

- Engine was hand tested by trying to perform game actions, such as playing a card, or placing a structure



```
> M, select where you would like to place your Road
> Please specify a tile x coordinate:
< 0
> Please specify a tile y coordinate:
< 0
(1) WEST: (-1, 0, 1)
(2) NORTH_WEST: (-1, 1, 0)
(3) SOUTH_WEST: (0, -1, 1)
(4) NORTH_EAST: (0, 1, -1)
(5) SOUTH_EAST: (1, -1, 0)
(6) EAST: (1, 0, -1)
> Please enter the number (e.g. '1') of the direction from the
center of the tile to the edge you would like to place a struct
ure on.
< 2
> Distributing resources.
> M received 1 brick cards.
> M received 1 lumber cards.
> M's turn:
> M: roll
> Player rolled a 7
> Distributing resources.
> M: aybabtu
> M: buy_card
> You received a Monopoly Card!
> M: 
```

Demo

Conclusion

- Start early and set regular, concrete deadlines as a team
- As a team, have a high-level understanding of your project’s design, but don’t be afraid to iterate and refactor the small(er) stuff

What Worked Well

- Slack / Trello / Github
- Weekly stand-ups

What We Would Have Changed

- Start implementation early!
- More unit tests for the engine