ProcGen & Tracery

Write Something That Writes Something

HELLO!

I am Austyn Hill aka The Lilac Llama

I am here because I am all procgen junkie
and I want to share the joy of casual creation with you.

You can find me at @TheLilacLlama or @ProcLlama

The Plan ...!

- ✘ First, let's define Proc Gen, and why we use it.
- ✘ Next – Tracery, and its nuts and bolts.
- ✘ Finally, Twitter Bots!

This may go a bit fast – but don't worry!

I'm putting all of these slides, resources and code online.
The github is in the meetup description, but you can also find it here.

https://github.com/LilacLlama/JSLou-1019

1.

What even IS ProcGen?

…

“

In computing, procedural generation is a method of creating data algorithmically as opposed to manually, typically through a combination of human-generated assets and algorithms coupled with computer-generated randomness and processing power.

– Wikipedia : Procedural Generation

Simpler, Please.

- ✘ ProcGen – Procedural Generation

- ✘ A Process for *Creation* of Content
  - ○ Uses <u>Curated Rules</u> or Building Blocks
  - ○ Generally a Sprinkle of <u>Randomization</u>
  - ○ And <u>an Engine</u> to make sense of it all

… generally code, that can generate *dynamic* content
that can cover a wide range of possibilities in place of static content.

Some Of My Favorite Things...

Dwarf Fortress

The deepest, most intricate simulation of a world that's ever been created.

Art Toy

A flower generator written by Kate Compton.

Cave of Qud

Is a science fantasy RPG & roguelike epic. It's set in a far future that's deeply simulated, richly cultured, and rife with sentient plants.

Spore

Spore is a 2008 life simulation real-time strategy God game developed by Maxis, published by Electronic Arts and designed by Will Wright.

Diablo

Is an action role-playing hack and slash video game developed by Blizzard.

Nested

A universe simulator.

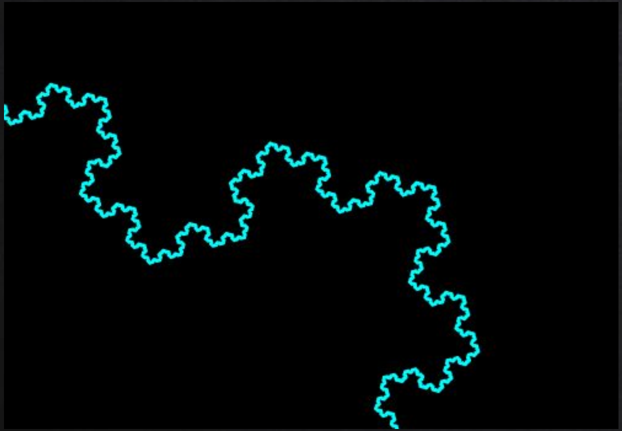
… but not JUST for games.

- ✘ If you can define it – you can generate it.
  - ○ Procedurally Generated Beethoven
  - ○ SkyKnit – Procedurally Generated Knitting
  - ○ Procedural Generated Terrain

… the problem is in the *defining.*

The Two Biggest Issues

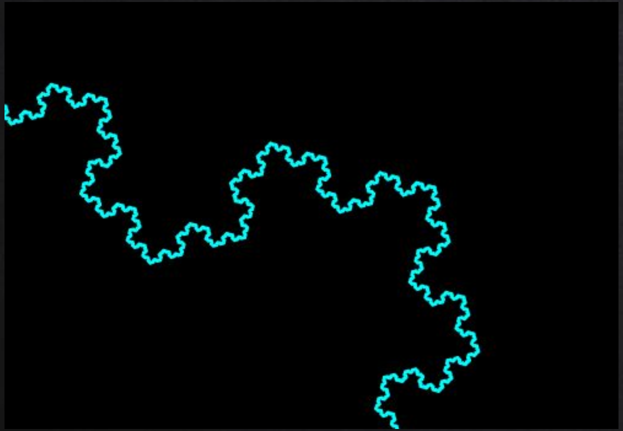
Scope



Oatmeal

The Two Biggest Issues

Scope



- ✘ So big.
- ✘ So intricate.
- ✘ So many edge cases.
- ✘ … you could go on forever.

The Two Biggest Issues

Oatmeal

- ✘ Lots of variation.
- ✘ .... but very little difference.
- ✘ .... and ultimately little impact.

... much like 1000 bowls of oatmeal.

The Solution?

- Start small.
  - Small parts.
  - Small pieces.
  - Small timeline.
- Fail fast
  - Try it.
  - Tweak it.
  - Expand it.

The Solution?

- ✘ Start small.
  - ○ Small parts.
  - ○ Small pieces.
  - ○ Small timeline.

- ✘ Fail fast
  - ○ Try it.
  - ○ Tweak it.
  - ○ Expand it.

… basically like writing MOST software.

Find your MVP and iterate iterate iterate!

2.

Tracery?

Light weight, author curated, text generation.

“

Tracery is a super-simple tool and language to generate text, by @galaxyKate.

The Basics

- ✘ Tracery
  - ○ Written by Kate Compton
  - ○ Originally in JavaScript*

- ✘ Requires an *author focused* JSON Grammar
- ✘ Expands rules to generate text**

* but 'ported to many other languages.

** remember, HTML is ultimately represented as text.

“

What does it mean to make an ‘author-focused’ generative text tool?

[P]otential authors for [a] generative tool to be creative persons [interested] in the expressivity of language and the aesthetics of prose. They may not self-identify as [‘programmers’] but they want to create generative text that is algorithmically combinatorial and surprising, [and still see] their authorial ‘voice’ in the finished text.

- Tracery: An Author-Focused Generative Text Tool

Grammars

- ✘ Written In JSON!
  - ○ Key -> Rule names
  - ○ Value -> Arrays of options

```
{
        “color”:[“blue”,”red”,“purple”]
}
```

Grammars

- ✘ Written In JSON!
  - ○ Key -> Rule names
  - ○ Value -> Arrays of options
- ✘ Expansions
  - ○ Select from options with #rule_name#

```
{
        "color":["blue","red","purple"],
        "origin":[
                "I like #color# cake.",
                "I like #color# hats.",
        ]
}
```

Grammars

Flatten #origin#:

1. Identify #origin# as expansion.
2. origin : I like #color# cake.
3. Identify #color# expansion.
4. color : blue
5. origin : I like blue cake.

```
{
        “color”:[“blue”,”red”,“purple”],
        “origin”:[
                “I like #color# cake.”,
                “I like #color# hats.”,
        ]
}
```

Grammars

- ✘ Written In JSON!
  - Key -> Rule names
  - Value -> Arrays of options
- ✘ Expansions
  - Select from options with #rule_name#
  - Combine them.

```
{
        “color”:[“blue”,”red”,“purple”],
        “origin”:[
            “I #feel# #color# #thing#.”
        ],
        “feel”:[“dislike”,”like”],
        “thing”:[“hats”,”cake”,”birds”]
}
```

Grammars

- ✘ Written In JSON!
  - ○ Key -> Rule names
  - ○ Value -> Arrays of options
- ✘ Expansions
  - ○ Select from options with #rule_name#
  - ○ Combine them.
  - ○ Nest them.

```
{
    “color”:[“blue”,”red”,“purple”],
    “origin”:[
        “I #feel# #obj#.”
    ],
    “feel”:[“dislike”,”like”],
    “obj”:[“#color# #thing#”],
    “thing”:[“hats”,”cake”,”birds”]
}
```

Grammars

- ✘ Written In JSON!
- ✘ Expansions
- ✘ Modifiers
  - English based
  - Use .modifier
  - Examples
    - .s -> pluralizes
    - .a -> adds article

```
{
        “color”:[“blue”,”red”,“purple”],
        “origin”:[
            “I #feel# #obj.s#.”
        ],
        “feel”:[“dislike”,”like”],
        “thing”:[“hat”,”cake”,”bird”]
}
```

Grammars

- ✘ Written In JSON!
- ✘ Expansions
- ✘ Modifiers
- ✘ Add Conditional Rules
  - [ruleName:#expansion#]
  - ‘Saves’ data of a sort.

```
{
  “color”:[“blue”,”red”,“purple”],
  “origin”:[
     “[colorSelect:#color#]
      I like #colorSelect# hats
      but not #colorSelect#
      bananas.”]
}
```

Some Helpful Resources

- ✘ Tutorials & Sandboxes
  - Crystal Code Palace Tutorial
  - Editor
  - Alison Parrish's Tutorial
  - Sculpting Generative Text
  - Learning Tracery
    - Examples & Definitions
    - Grammars & More Grammars

3.

Bot Making

… a quick way to play with proc gen text

“

An autonomous program on a network (especially the Internet) that can interact with computer systems or users.

Cheap Bots Done Quick

- ✘ Based on Tracery
- ✘ Requires Twitter Auth
- ✘ Simple Grammars / Reply rules
- ✘ .... that’s it.

... literally the hardest part is deciding what you want to write.

WORKSHOP TIME!

- ✘ Next are some Example 'Bot's – but set up as codepen templates.
- ✘ You can yoink the grammar straight from the bitly link.
- ✘ Or you can tweak it first!

Compliment Bot

https://bit.ly/2Nq8QG3

🌴🐪🌴

Emoji Story Bot

https://bit.ly/2JBk7Ct

FACE Bot

https://bit.ly/2N1HAPa

Next Steps...

Cheap Bots Done Quick is **AWESOME** ... but also simple.
Here are some ways to expand your ‘bot EVEN FARTHER!

- ✘ Do it w/ Glitch!
  - Fancier interactions
  - Better JS control
- ✘ Consider Bottery
  - Inspired by Tracery / similar elements
  - But with more ‘state machine’ bits.
- ✘ Read up on BotWiki
  - Lots of options & languages!

Other Helpful Resources

- Grammar Assistant
- Grammar Elements
- Corpora
  - Dariusk / Corpora
  - NLP Corpora
  - Project Gutenberg

- Shiny Example Bots
  - @thetinygallery
  - @TinyAdv
  - @ColorSchemez
  - @softlandscapes

THANKS!

Any questions?

You can find me at
@ProcLlama @TheLilacLlama
TheLilacLlama@gmail.com

https://github.com/LilacLlama/JSLou-1019