Procedural Generation of Story-Driven Maps

By
Elizabeth Matthews
Brian Malloy
Motivation

Why is procedural generation a good thing?

What's missing?
Overall Concept

- Locations of interest
- Restrictions between Locations
- Connected graph
Initialization

- World data stored in XML
  - Nodes
    - Towns, caves, generators
  - Restrictions
- Python 2.6
  - Mini-dom
  - Pygame
<town>
  <name>Cave of Figaro</name>
  <areas>
    <R>mountain</R>
    <R>grass</R>
  </areas>
  <restriction>
    <type>town</type>
    <name>Town of South Figaro</name>
    <min>10</min>
    <max>20</max>
    <typed>T</typed>
  </restriction>
  ...
</town>
Simulation Phase

- Finding locations for the nodes
- Each node made into physical body
- Restrictions redefined as springs
- Place nodes in space
- Springs provide forces on nodes
Floodfill Phase

- Nodes have terrain defined for nearby areas
- Initialize
  - Nodes, generators
  - Edges
  - “Filler”
- Use edges to flood unfilled areas
Floodfill Phase

- Unforeseen Problems
- Isolated areas
- Uneven flood
Generations

- Create a world in pieces
- First generation
  - Limited to “walkable” areas
- Second generation
  - Nodes
    - first generations
    - collisions
  - Restrictions
    - “unwalkable”
Future Work and Applications

- Third generation
- Forced mode
- Real time or Offline
- 3D from 2D
- Personal map generation
Procedural Generation of Story-Driven Maps

By
Elizabeth Matthews
Brian Malloy
Thinking about this problem, we came up with an idea for a system that would respond to the rigid requirements from a story-driven game, while still allowing for unique map creation.
Overall Concept

- Locations of interest
- Restrictions between Locations
- Connected graph

- “Nodes” can be any location of interest

- Minimum and maximum distance
- “Typed” requiring a specific type of terrain

- Only direct connections.

- Not the same thing as planar graphs
World data is stored in XML format.

Python minidom used

Once built, we can use these values to build a physical structure to estimate the locations of the nodes.
<town>
  <name>Cave of Figaro</name>
  <areas>
    <R>mountain</R>
    <R>grass</R>
  </areas>
  <restriction>
    <type>town</type>
    <name>Town of South Figaro</name>
    <min>10</min>
    <max>20</max>
    <typed>T</typed>
  </restriction>
  ...
</town>
Simulation Phase

- Finding locations for the nodes
- Each node made into physical body
- Restrictions redefined as springs
- Place nodes in space
- Springs provide forces on nodes

- A graph with connected nodes needs to be settled on a 2D plane
- “rest-length” set to random value
- Between min and max
- Random x,y within the space
- Prefered rest length
- Reactive stretching
Floodfill Phase

- Nodes have terrain defined for nearby areas
- Initialize
  - Nodes, generators
  - Edges
  - “Filler”
- Use edges to flood unfilled areas
Floodfill Phase

- Unforseen Problems
- Isolated areas
- Uneven flood

- “Walkable” must be ensured
- Create a “safe” buffer on “typed” restrictions
- Areas must look right
- Pick first choice by spawner

Now that a small subsection of a map has been generated, we take a slightly recursive step back to generate larger chunks of the map.
Generations

- Create a world in pieces
- First generation
  - Limited to “walkable” areas
- Second generation
  - Nodes
    - first generations
    - collisions
  - Restrictions
    - “unwalkable”

What we've created is a small subsection of a continent. A place where every location of interest is accessible from the others. Generations handle continuing the creation of a world. Collision is so that map pieces don't override previously created subsections.
Talk about third generation here
• Work in progress
  • Collision detection
  • Generator placement
  • Hardcoded parts
• Extras
  • Manual mode
  • Two options for area definition for nodes
  • Identify mode
Future Work and Applications

- Third generation
- Forced mode
- Real time or Offline
- 3D from 2D
- Personal map generation