Procedural Generation of Story-Driven Maps

By Elizabeth Matthews
Motivation

- Sequence of locations
  - Story driven games have location restrictions
  - Random can't guarantee a path from A to B

- Exploration
  - The story is set
  - Replayability resides in exploring the map

- Connected graph
What is Procedural Generation?

- A procedure to generate content.
  - Randomized
  - Consistent
  - Fast and “close enough”
- Offline
  - Pre-generation allows for complexity/size
- Real-Time
  - No/Minimal storage required
Procedural Examples

- Perlin Noise
- L-Systems
- Use in games
  - Diablo
  - Left 4 Dead
  - Canabalt
Building Phase

- Python 2.6
- Nodes
  - Points of interest
  - Desired area traits are defined
- Restrictions
  - Between nodes
  - Min/Max values
  - “Restricted”
Simulation Phase, I

• Finding locations for the nodes
  – A graph with connected nodes needs to be settled on a 2D plane

• Similar to planar graphs
Planar Graphs, I

- Graphs that can be placed in a 2D plane
  - No edges can cross
- Kuratowski's and Wagner's theorems
  - Subgraphs, Minors
  - Slow
Planar Graphs, II

- Fast approximations:
  - Theorem 1. If $v \geq 3$ then $e \leq 3v - 6$;
  - Theorem 2. If $v > 3$ and there are no cycles of length 3, then $e \leq 2v - 4$
- Sparse graphs
Planar Graphs, III

- Vehicle routing
- Circuit layouts
- Telecommunications
Planar Graphs, IV

- Too rigid
  - Restrictions are reactive
  - Edges (restrictions) can cross
Simulation Phase, II

- Physics Engine!
  - Pygame, visualization
  - Pymunk, simulation
    - Pymunk, Chipmunk
Simulation Phase, III

- Each node made into physical body
- Restrictions redefined as springs
  - “rest-length” set to random value
  - Between min and max
- Read in from XML file
  - Python minidom
  - XML is created from CSV
Simulation Phase, IV

- Place nodes in space
  - Random x,y within the space
- Springs provide forces on nodes
  - Preferred rest length
  - Reactive stretching
- Friction
- Wait for nodes to settle
  - Total velocity over all nodes within a threshold
Floodfill Phase, I

- Nodes have terrain defined for nearby areas
- Fill the map
- Similar to Voronoi Diagrams
Voronoi Diagrams, I

- A partition of 2D (or 3D) space
- Based on a set of points
- Areas reflect nearest point
Voronoi Diagrams, II

- $F_1(x)$
  - Distance to closest point

- $F_2(x)$
  - Distance to second closest

- $F_n(x)$
  - Distance to $N$th closest
Voronoi Diagrams, III

- $F_2(x) - F_1(x)$
- Other distance functions produce different results
- Worley Noise
Voronoi Diagrams, IV

- Nearest neighbor
- Robot AI navigation
- Procedural generation
Voronoi Diagrams, V
Voronoi Diagrams, IV

- Too regular
  - Areas too rigid
  - Randomized areas
Floodfill Phase, II

- Initialize
  - Towns, generators
  - Edges
  - “Filler”
Floodfill Phase, III

- Use edges to flood unfilled areas
  - Pick edge
  - Pick direction
  - Flood with choice's terrain
  - Cannot override already filled areas
Floodfill Phase, IV

- Unforeseen Problems
- Isolated areas
  - Create a “safe” buffer on “restricted” lines
- Uneven flood
  - Pick first choice by spawner
Generations, I

- First generation
  - Limited to “walkable” areas

- Second generation
  - Nodes are redefined to first generations
  - Nodes have collision
  - Restrictions now must be “unwalkable”
  - Set terrain generation based on restrictions
Generations, II
Demo

- Second generation still 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
  - Nodes are second generation
- Forced mode
- Real time or Offline
- 3D from 2D
- Personal map generation
- C++ for speed
Questions?