

Ogun: Spatial Layout Generation via Sequential Logit Dynamics on Potential Games

Elias Vahlberg

March 2026

Abstract

We present Ogun, a deterministic spatial layout algorithm that places nodes and routes edges on a 2D grid by modeling placement as a potential game solved via sequential logit dynamics. Agents arrive in a fixed order, evaluate a potential function Φ encoding attraction, repulsion, and configurable kernel terms over every grid cell, then commit irrevocably to a position sampled from the Boltzmann distribution $P(p) \propto \exp(\beta \cdot \Phi(p))$. Edges are routed during placement using congestion-aware Dijkstra and refined post-placement via PathFinder negotiation (rip-up-and-reroute). The inverse temperature β acts as a single control parameter governing output character, from near-random ($\beta \rightarrow 0$) to near-optimal ($\beta \rightarrow \infty$). We describe the algorithm, analyze its complexity, and demonstrate its application to procedural city generation.

Keywords: procedural generation, potential games, logit dynamics, spatial layout, pathfinding, negotiation routing

1 Introduction

Procedural generation of structured 2D layouts (cities, dungeons, circuit boards) requires solving two coupled problems simultaneously: *placement* (where to put things) and *routing* (how to connect them). Classical approaches treat these as separate optimization passes: force-directed placement followed by maze routing [2]. This decoupling loses information; placement decisions made without routing awareness produce layouts that are difficult or impossible to route efficiently.

Ogun couples placement and routing into a single sequential process. Each agent (node) observes the current state, including routes already laid, before choosing its position. The placement decision is formalized as a potential game where the potential function Φ encodes both local quality (overlap, repulsion, attraction to neighbors) and global structure (boundary affinity, terrain compatibility). The Boltzmann sampling mechanism provides a principled interpolation between greedy optimization and stochastic exploration, controlled by a single parameter β .

The algorithm draws on two traditions: *logit dynamics* from evolutionary game theory [1], where agents update strategies with noise proportional to $1/\beta$; and *negotiation-based routing* from electronic design automation [2], where competing routes iteratively resolve conflicts through congestion penalties. The combination produces layouts that are structured yet organic, well-suited to procedural content generation for games.

The name *Ogun* comes from the Yoruba deity of iron, metalwork, and pathfinding, a god who cleared the first roads through primordial wilderness.

2 Problem Formulation

2.1 Definitions

Let $G = (V, E)$ be a weighted undirected graph where each node $v_i \in V$ has a footprint radius $r_i \in \mathbb{N}$ and each edge $e_k = (v_i, v_j, w_k)$ has a weight $w_k > 0$ encoding connection importance.

Let $S = (W, H, \mathcal{O})$ be a spatial domain: a $W \times H$ integer grid with obstacle set $\mathcal{O} \subset [0, W) \times [0, H)$. Optionally, S carries a routing cost grid $\rho : [0, W) \times [0, H) \rightarrow \mathbb{R}^+$ (default $\rho \equiv 1$).

2.2 Output

A *layout* L consists of:

- A position map $\pi : V' \rightarrow S$ for placed nodes $V' \subseteq V$
- A path map $\gamma : E' \rightarrow S^*$ for routed edges $E' \subseteq E$
- A score breakdown $(s_{\text{pe}}, s_{\text{acc}}, s_{\text{cong}}, s_{\text{vr}}) \in [0, 1]^4$

2.3 Objective

Maximize the composite score $s = \frac{1}{4}(s_{\text{pe}} + s_{\text{acc}} + s_{\text{cong}} + s_{\text{vr}})$ subject to:

1. No two node footprints overlap
2. No node footprint intersects an obstacle
3. All paths are 4-connected and avoid obstacles and footprints (except at endpoints)

The parameter β does not appear in the objective; it controls the *stochasticity* of the search, not the target.

3 Algorithm Overview

Ogun proceeds in two phases. Phase 1 places nodes sequentially with congestion-aware routing after each placement. Phase 2 refines all routes via PathFinder negotiation.

Algorithm 1: Ogun: Sequential Logit Dynamics with Negotiated Routing

Input: Graph $G = (V, E)$, Space S , Config $(\beta, \text{seed}, K, \rho)$ **Output:** Layout L

```
1 rng  $\leftarrow$  ChaCha8(seed);
2 foreach node  $v_i$  in arrival order do
3   foreach valid position  $p \in S$  do
4      $u_i(p) \leftarrow \Phi(p, v_i, \text{state});$  // Evaluate potential
5   end
6    $p^* \sim \text{Boltzmann}(\{u_i(p)\}, \beta, \text{rng});$  // Sample
7   Place  $v_i$  at  $p^*$ ; block footprint;
8   foreach edge  $(v_i, v_j)$  where  $v_j$  is placed do
9      $\gamma_{ij} \leftarrow \text{DijkstraCongestion}(p^*, \pi(v_j), S);$ 
10  end
11 end
12 for  $t = 1$  to  $T_{\text{neg}}$  do
13   foreach edge  $e_k$  sorted by weight descending do
14     Rip up  $\gamma_k$ ; reroute via Dijkstra with updated congestion;
15   end
16   if no shared cells then
17     break;
18   end
19   Update history penalties on congested cells;
20 end
21 Score layout and return  $L$ ;
```

4 Potential Function

The potential $\Phi(p, v_i, \text{state})$ decomposes into four terms:

$$\Phi(p, v_i) = - \underbrace{\sum_{j \in N(i)} w_{ij} \cdot d(p, \pi(v_j))}_{\text{attraction}} + \underbrace{\sum_{j \in \text{placed}} \frac{k_r \cdot m_{ij}}{d^2(p, \pi(v_j))}}_{\text{repulsion}} + \underbrace{\Phi_{\text{boundary}}(p, v_i)}_{\text{affinity}} + \underbrace{B(p)}_{\text{cell bonus}} \quad (1)$$

where $N(i)$ is the set of placed neighbors of v_i , k_r is the global repulsion constant, m_{ij} is a per-pair repulsion multiplier (default 1), and $d(\cdot, \cdot)$ is Euclidean distance.

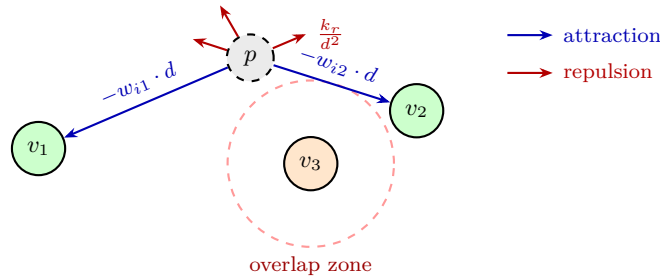


Figure 1: Forces acting on a candidate position p for a new node v_i . Blue arrows show attraction toward connected, already-placed neighbors (v_1, v_2). Red arrows show repulsion pushing p away from all placed nodes. If p falls within the overlap zone of any placed node (dashed circle around v_3), the position is rejected.

Overlap rejection. If $d(p, \pi(v_j)) < r_i + r_j$ for any placed node v_j , or if $p \in \mathcal{O}$, the position is rejected ($\Phi = \perp$).

Repulsion cutoff. Contributions where $d^2 > 100k_r$ are skipped, bounding the repulsion pass at $O(n_{\text{nearby}})$ rather than $O(n)$.

Boundary affinity. For nodes with affinity $a_i \neq 0$:

$$\Phi_{\text{boundary}}(p, v_i) = a_i \cdot \left(1 - \frac{d_{\text{edge}}(p)}{\lfloor \min(W, H)/2 \rfloor}\right) \quad (2)$$

where $d_{\text{edge}}(p) = \min(p_x, W-1-p_x, p_y, H-1-p_y)$.

Cell bonus. An optional grid $B : S \rightarrow \mathbb{R}$ adds per-cell utility for terrain compatibility or zoning preferences.

5 Boltzmann Sampling

Given candidate utilities $\{u_1, \dots, u_m\}$, the agent selects position p_j with probability:

$$P(p_j) = \frac{\exp(\beta \cdot u_j)}{\sum_{k=1}^m \exp(\beta \cdot u_k)} \quad (3)$$

The log-sum-exp trick prevents overflow: subtract $u_{\max} = \max_k u_k$ before exponentiation. Sampling uses inverse CDF with a single uniform draw, requiring no weight vector allocation (two-pass streaming).

As $\beta \rightarrow \infty$, sampling converges to $\arg \max$; as $\beta \rightarrow 0$, sampling becomes uniform. This provides a smooth interpolation between greedy placement and random exploration.

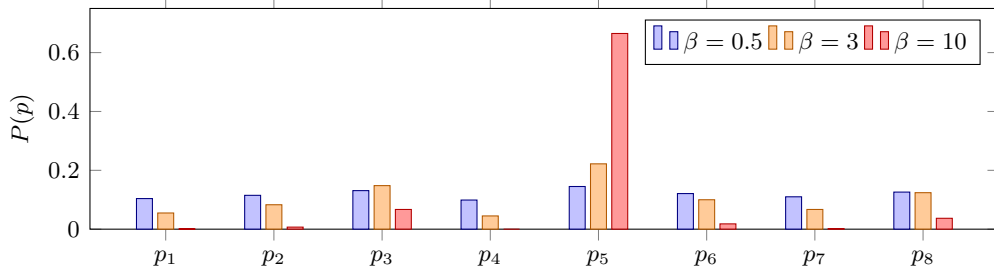


Figure 2: Effect of inverse temperature β on position selection probability for eight candidate positions with varying utility. At low β , the distribution is nearly uniform (exploratory). At high β , probability concentrates on the highest-utility position p_5 (greedy). Intermediate β balances exploration and exploitation.

6 Routing

6.1 Phase 1: Congestion-Aware Dijkstra

After each node placement, edges to already-placed neighbors are routed via Dijkstra on the 4-connected grid. The edge cost at cell p is:

$$C(p) = \rho(p) \cdot (1 + h(p)) \cdot (1 + s(p)) \quad (4)$$

where $\rho(p)$ is the base routing cost, $h(p)$ is accumulated history penalty, and $s(p)$ is the current sharing count (number of routes using p).

Cells blocked by node footprints are impassable except at route endpoints.

6.2 Phase 2: PathFinder Negotiation

After all nodes are placed, routes are iteratively ripped up and rerouted to resolve congestion, following the PathFinder algorithm [2]:

1. For each edge (sorted by weight descending): remove its path from the sharing grid, reroute via Dijkstra with current congestion costs, add the new path back.
2. If no cell has $s(p) > 1$, terminate (convergence).
3. Otherwise, increment history: $h(p) += \delta$ for all p where $s(p) > 1$.
4. Repeat for up to T_{neg} iterations.

Higher-weight edges are rerouted first, giving important connections priority over minor ones. The history penalty ensures that persistently congested cells become increasingly expensive, forcing routes to find alternatives.

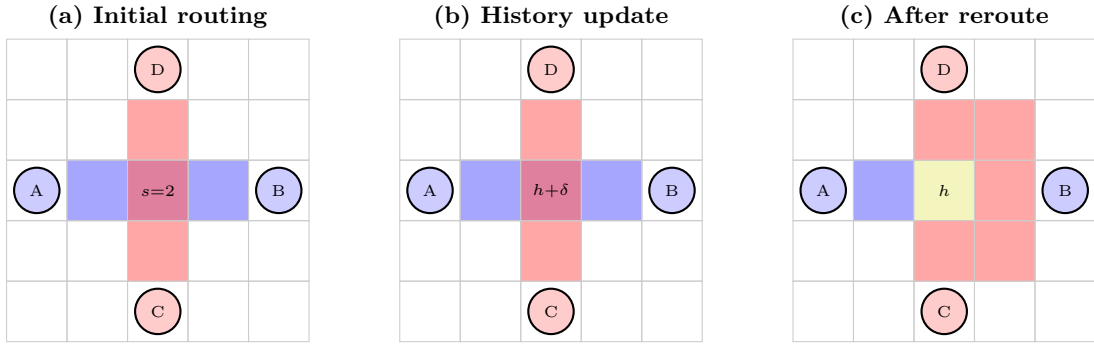


Figure 3: PathFinder negotiation resolving a routing conflict. (a) Two routes (blue: A→B, red: C→D) share a cell ($s = 2$). (b) The shared cell receives a history penalty $h + \delta$, increasing its cost. (c) On the next iteration, the lower-weight route (red) detours around the now-expensive cell, eliminating the conflict.

7 Scoring

The composite score combines four normalized metrics:

Metric	Definition	Ideal
Path efficiency s_{pe}	$\frac{1}{ E' } \sum_{e \in E'} \frac{d_{\text{manhattan}}(e)}{ \gamma(e) }$	1.0
Accessibility s_{acc}	$ E' / E $	1.0
Congestion s_{cong}	$1/\max_p s(p)$	1.0
Void ratio s_{vr}	$1 - \frac{ r_{\text{void}} - 0.2 }{0.2}$	1.0

Table 1: Scoring metrics. Each is in $[0, 1]$; the composite is their mean.

The void ratio metric penalizes deviation from $\sim 20\%$ empty space, encoding the observation that believable layouts are neither packed solid nor mostly empty.

8 Implementation

Ogun is implemented in Rust (edition 2024, MSRV 1.85) with the following performance-critical design choices:

Struct-of-arrays (SoA) layout. Placed node coordinates are stored as separate `Vec<f32>` arrays (x, y, radius) rather than an array of structs, enabling SIMD auto-vectorization of the repulsion loop.

Parallel evaluation. For large grids ($W \times H \times n_{\text{placed}} \geq 500,000$), utility evaluation is parallelized across rows using Rayon.

Buffer reuse. Dijkstra distance/predecessor arrays and BFS queues are allocated once and reused across all routing calls.

Determinism. All randomness flows through a single `ChaCha8Rng` seeded from the configuration. Same input = same output.

8.1 Benchmarks

Nodes	Grid	Time	Speedup vs. naive
50	100×100	44 ms	$3.2\times$
100	250×110	323 ms	$5.0\times$
200	250×250	1.28 s	$9.5\times$
500	500×500	11.3 s	$24.8\times$

Table 2: Release-mode benchmarks. Naive baseline uses AoS layout, no cutoff, no parallelism, no buffer reuse.

9 Application: Procedural City Generation

Figure 4 shows a city layout generated by `Oku`¹, a procedural city generator built on `Ogun`. Graph nodes represent buildings (residential, commercial, military, religious); edges represent infrastructure demands (roads, utilities). The `boundary_affinity` kernel places military structures near the perimeter, while `cell_bonus` encodes zoning preferences. Routing costs create natural road hierarchy: main arteries through low-cost corridors, minor streets through higher-cost residential areas.

10 Complexity Analysis

Let $n = |V|$, $m = |E|$, $A = W \times H$.

Phase	Time	Space
Utility evaluation (per node)	$O(A \cdot n)$	$O(A)$
Boltzmann sampling (per node)	$O(A)$	$O(1)$
Dijkstra routing (per edge)	$O(A \log A)$	$O(A)$
Phase 1 total	$O(n \cdot A \cdot n + m \cdot A \log A)$	$O(A + n + m)$
Phase 2 (T iterations)	$O(T \cdot m \cdot A \log A)$	$O(A + m)$
Scoring	$O(m \cdot A)$	$O(A)$

Table 3: Asymptotic complexity. $A = W \times H$ is the grid area.

¹<https://github.com/EliasVahlberg/oku>

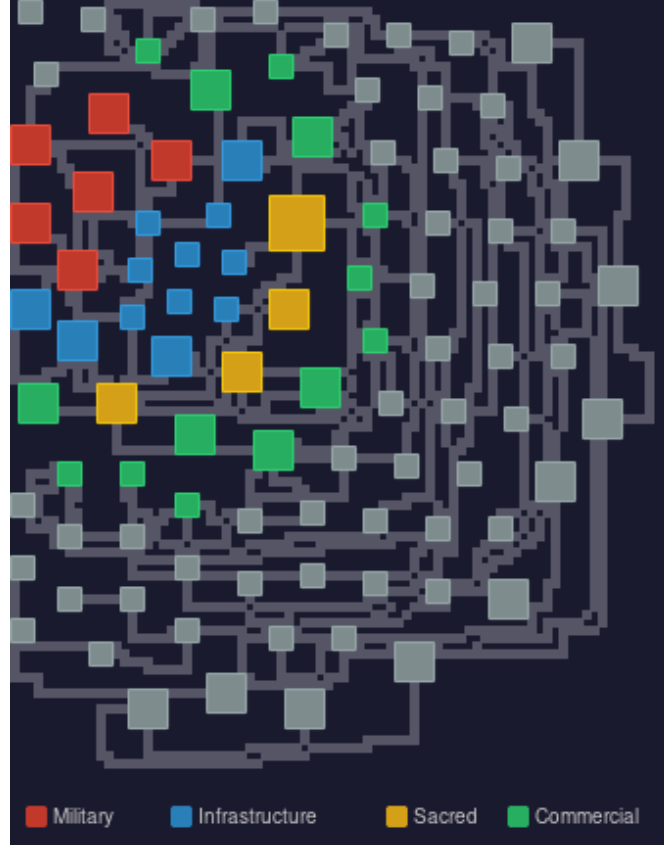


Figure 4: A procedural city layout generated by Oku using the Ogun algorithm. Colored rectangles represent buildings of different types; gray paths are routed roads. The layout exhibits emergent clustering and road hierarchy from the potential function and negotiation routing.

The dominant cost is utility evaluation: $O(n^2A)$ total across all nodes. With the repulsion cutoff and Rayon parallelism, the practical constant is small; the 500-node benchmark completes in 11.3s on a 500×500 grid.

11 Conclusion

Ogun demonstrates that sequential logit dynamics provide an effective framework for coupled placement-and-routing on 2D grids. The potential game formulation gives a clean decomposition of placement quality into interpretable terms (attraction, repulsion, boundary affinity, cell bonus), while the Boltzmann parameter β provides principled control over the optimization-exploration tradeoff. PathFinder negotiation routing, adapted from EDA, produces emergent road hierarchy without explicit priority assignment.

The algorithm is deterministic, domain-agnostic, and thread-safe. Its application to procedural city generation via the Oku system demonstrates that EDA-derived techniques transfer effectively to game content generation.

Future work includes hierarchical generation (running Ogun at district, block, and building scales), a target convergence loop that adjusts β to reach a desired composite score, and functional erosion, a cascading degradation model that transforms optimized layouts into believable ruins.

References

- [1] L. E. Blume, “The statistical mechanics of strategic interaction,” *Games and Economic Behavior*, vol. 5, no. 3, pp. 387–424, 1993.
- [2] L. McMurchie and C. Ebeling, “PathFinder: A negotiation-based performance-driven router for FPGAs,” in *Proc. ACM/SIGDA FPGA*, pp. 111–117, 1995.
- [3] D. Monderer and L. S. Shapley, “Potential games,” *Games and Economic Behavior*, vol. 14, no. 1, pp. 124–143, 1996.
- [4] C. Y. Lee, “An algorithm for path connections and its applications,” *IRE Trans. Electronic Computers*, vol. EC-10, no. 3, pp. 346–365, 1961.
- [5] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-based procedural content generation: A taxonomy and survey,” *IEEE Trans. Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.
- [6] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural Content Generation in Games*. Springer, 2016.