# Maximizing Territory in the Voronoi Game: A Graph-Based Influence Strategy for Store Placement

## Abstract

In this project, I modeled a real-world competition between companies choosing store locations, where each store “claims” nearby districts, and the goal is to maximize total customer value captured. Using graphs and shortest-path distances as the core CS concepts, I implemented and compared a territory-based Influence Player, which evaluates each move by the marginal value of vertices it would newly bring under that player’s control, with the Random and Greedy Player. I then ran large-scale simulations on random graphs with 100 vertices, measuring win rates and score margins against both a purely random player and a simple greedy player that always chooses the highest-value free vertex. The influence-based algorithm won 100% of games against the random player and 99.5% of games against the greedy player, usually by large margins, showing that optimizing for territorial control is much more effective than either naive randomness or local value greed. Overall, my main takeaway is that in Voronoi-style games (and analogous real-world location problems), reasoning about the structure of the entire graph and the regions induced by each move is crucial for good performance.

## Results

- **The Algorithm**

The core idea of my algorithm is not to think of moves as picking valuable vertices, but instead to think of them as placing a store that pulls in as much new territory as possible. That’s why I named my class VoronoiInfluencePlayer, which is based on how much territory a move influences or captures.

On each turn, my Influence algorithm does:

- Loop over every candidate vertex that does not already have a token;
- For each candidate c, it looks at every vertex v in the graph and compares the distance form v to c, and the distance from v to its current closest token, if there is one;
- If placing a token at c would make c the strictly closest token to v, then v would switch its allegiance to the player who plays c.

However, I only want to coin new gains, not reshuffling stuff I already own. So I check who currently owns v using graph.getCurrentOwner(v). I only add graph.getValue(v) to the candidate’s score if v is currently unowned or owned by the other player, and the new token at c would become the closest token to v. Thus the definition of the score for a candidate vertex c is the sum of values of all vertices that are not currently mine but would becom eclosest to a token owned by me if I place a store at c. Then I choose the candidate with the highest score, and break ties by preferring the candidate with a higher intrinsic vertex value, so if two moves pull in the same total new value, I choose the one that also sits on a more valuable vertex itself. Because the VoronoiGraph precomputes all-pairs shortest path distances using Floyd-Warshall, each distance lookup (graph.getDistance(u, v) is O(1), which makes this style of influence scoring feasible within the 0.5 seconds per move limit.

- **Design Process**

I built up to it, starting from the Random and Greedy algorithms. The Random algorithm simply picks a random free vertex each turn. It totally ignores the vertex values, the distances, and which vertices are already controlled. For the Greedy algorithm, it picks the highest-value vertex that doesn’t have a

token. However, the player who picks the two highest-value vertices can still lose the game. Then, the total value of the region that ends up closest to the stores is important. The Greedy algorithm ignores edges, graph structure, and distances, and an isolated 90-value vertex in a corner might be worse than a 40-value vertex that can pull in a huge cluster of medium-value neighbors.
My first version of the Influence algorithm adds value(v) to the candidate's score if distance(v,c) < distance(v, closestToken(v)) for each candidate c and each vertex v. This one overcounts because, if a vertex v is already mine, and I place another token that's even closer, the total score for my player doesn't change. The nearest store label just changes from one of my stores to another, and the scoring treated this as a gain, which is wrong logically.
Then I have the second version, which is the current algorithm. I integrated player ownership into the scoring. For each vertex v, I check the distance to the candidate, the current closest token and its distance, and the current owner. Then I only count value(v) toward the candidate's score if the candidate would become closest, and it is unclaimed or is owned by the opponent.

- **Experiment**

To evaluate the performance of my VoronoiInfluencePlayer algorithm, I wrote a separate driver class ComparePlayers that runs batch simulations between two player algorithms without visualization and records summary statistics to a text file.
Each trial proceeds as follows:

1. I generate a new random Voronoi graph using the constructor new VoronoiGraph(n, density) with n=100 vertices and edge density 0.1, matching the project specification.
2. Each vertex is assigned an independent random value in [0, 100], and edges have random weights in [1, 2], as in the provided framework.
3. I instantiate two players on this graph: algorithm A and algorithm B. I then play two games per graph: A as player 0 (moves first), B as player 1 (moves second), B as player 0, A as player 1.
4. In each game, both players take exactly 10 turns, alternating moves. Each call to chooseVertex is executed in a separate thread with a time limit of 500 ms; if a player times out, their move for that turn is skipped.
5. After all moves are played and tokens are placed, I query VoronoiGraph.playerValues() to obtain the total vertex value controlled by each player. The driver then maps these totals back to A and B, depending on the ordering, and updates the statistics.

For each matchup (Influence vs Random, Influence vs Greedy), I repeat this on 100 different random graphs, with both orderings on each graph. This yields a total of 200 games per matchup. The driver records the number of wins, losses, and ties for each algorithm, the average score for each algorithm across games, the average score margin (A − B), and how often algorithm A wins as first player vs as second player. The results are written to compare_results.txt under the src directory.

- **Hypothesis**

First, I expected the influence-based algorithm to be consistently better than random, because it explicitly evaluates how much territory (by total vertex value) a candidate store location can pull in, while the random player completely ignores values and distances. Second, I expected the influence-based algorithm to be usually better than greedy, because greedy only looks at the value of the chosen vertex itself, not the full Voronoi region that ends up controlled by that choice. Since my algorithm tries to maximize new captured value (vertices that newly switch from unclaimed or opponent-owned to my control), it should be closer to optimizing the actual scoring rule of the game.

- **Table of Results**

Table 1. Performance of the influence-based algorithm vs. the random player on 200 games (100 graphs * 2 orderings).
Each entry is averaged over 200 games, consisting of 100 random graphs with 100 vertices and edge density 0.1, with each graph played twice (once with Influence moving first, once with Random moving first). “Avg score A/B” is the average total vertex value controlled by each algorithm; “Avg margin” is the average difference (Influence − Random); “win rate” is the fraction of games won.

| **Metric** | **Value** |
|---|---|
| Games | 200 |
| Influence (A) wins | 200 |
| Random (B) wins | 0 |
| Ties | 0 |
| Avg score – Influence (A) | 3410.720 |
| Avg score – Random (B) | 1504.690 |
| Avg margin (A − B) | 1906.030 |
| Influence win rate | 1.000 |
| Random win rate | 0.000 |
| Tie rate | 0.000 |

Across 200 games, the influence-based player wins 100% of the time against the random player, with no ties. Its average total score is about 3411, compared to roughly 1505 for the random player, so the average margin of victory is over 1900 points. This means that, on a typical random graph under the project’s parameters, the influence strategy captures more than twice as much vertex value as random placement. Because turn order is alternated for each graph, this near-perfect dominance indicates that the advantage comes from the algorithm’s ability to choose strong locations rather than from simply moving first.

Table 2. Performance of the influence-based algorithm versus the greedy player on 200 games (100 graphs * 2 orderings).
Setup is identical to Table 1: 200 games total (100 random graphs * 2 orderings), 10 turns per player, and a 500 ms time limit per move. The greedy player always selects the highest-value free vertex; the influence player selects the vertex that maximizes newly captured value.

| **Metric** | **Value** |
|---|---|
| Games | 200 |
| Influence (A) wins | 199 |
| Greedy (B) wins | 1 |

| **Metric** | **Value** |
|---|---|
| Ties | 0 |
| Avg score – Influence (A) | 3107.730 |
| Avg score – Greedy (B) | 1870.960 |
| Avg margin (A − B) | 1236.770 |
| Influence win rate | 0.995 |
| Greedy win rate | 0.005 |
| Tie rate | 0.000 |

When facing the greedy player, the influence-based algorithm wins 199 out of 200 games, losing only once and never tying. On average, Influence scores about 3108 points while Greedy scores about 1871, for an average margin of roughly 1237 points. The win rate stays extremely high even when the influence player moves second for half of the games, and the large positive margin shows that it does not just edge out small victories but consistently captures significantly more territory. The fact that a global, territory-based heuristic beats a vertex-value-greedy strategy this decisively suggests that controlling high-value regions is much more important than simply sitting on the single highest-value vertices.

- **Synthesis**

These results strongly confirm the hypothesis that a Voronoi-style influence heuristic is both consistently better than random and usually better than greedy in this game. Against the random player, the influence algorithm never loses: across 200 games, it wins every single time and more than doubles the opponent's average score. This aligns with the intuition that random placement completely ignores structure—edge weights, distances, and vertex values—while the influence player explicitly evaluates how much new territory each potential move will capture. In the "store placement" story, the influence algorithm is behaving like a retailer that actually studies the map and customer distribution, versus one that just throws darts at a wall.

The comparison with the greedy player is more revealing, because Greedy at least uses vertex values and feels "smart" at first glance. However, the data shows that a locally greedy strategy is still not enough: the influence algorithm wins 199 out of 200 games and outperforms Greedy by more than 1200 points on average. This happens because Greedy focuses on the value of the store location itself, often picking isolated or poorly positioned high-value vertices, while the influence player prefers vertices that claim or steal large, high-value regions—even if the chosen vertex's own value is smaller. In real-world terms, this is the difference between picking the single richest district versus picking a slightly less glamorous intersection that gives you access to many surrounding neighborhoods.

Together, the two tables show a consistent pattern: algorithms that are aware of the global structure of the graph and the resulting Voronoi regions dramatically outperform those that only consider local information (vertex value) or no information (random). This supports the broader takeaway that for location-allocation problems like store placement, cell tower positioning, or facility location, it is not enough to look at individual points in isolation. Instead, good strategies must reason about how each decision reshapes the entire territory—who becomes closer to you than to your competitor and how much value those regions contain.

## Acknowledgements

I would like to thank Professor Harper and Professor Al Madi for providing the project template and guidance site, as well as the TAs for guiding me through the debugging process. I also appreciate that my classmates help me debug. Thanks to Oracle and W3Schools for the detailed Java documentation and Wikipedia pages (Greedy Algorithm, Voronoi diagram), which helped me quickly grasp the ideas for designing the algorithm in the project.

---

## Extensions:

- **What**

For my extension, I implemented an additional player algorithm called VoronoiNeighborhoodPlayer. Instead of looking at global territory like the influence-based player, this algorithm uses a simpler local heuristic: for each candidate vertex, it sums the values of all vertices within a fixed shortest-path radius (I used radius = 2.0) and picks the candidate whose “neighborhood” has the highest total value, breaking ties by intrinsic vertex value. The intuition was to create a strategy that favors dense local clusters of value without explicitly reasoning about current ownership or Voronoi regions. I wanted to see how much performance I could get out of a purely local density heuristic compared to my more sophisticated influence-based player and the simpler baselines.

- **Outcome**

Table 3. Performance of Neighborhood player versus Random player in the Voronoi game on graphs.
Each entry is averaged over 200 games (100 random graphs × 2 orderings) on VoronoiGraph(n = 100, density = 0.1). “Avg score A/B” is average total vertex value; “Avg margin” is Neighborhood − Random.

| **Metric** | **Value** |
|---|---|
| Games | 200 |
| Neighborhood (A) wins | 144 |
| Random (B) wins | 56 |
| Ties | 0 |
| Avg score – Neighborhood (A) | 2648.465 |

| **Metric** | **Value** |
|---|---|
| Avg score – Random (B) | 2316.305 |
| Avg margin (A − B) | 332.160 |
| Neighborhood win rate | 0.720 |
| Random win rate | 0.280 |
| Tie rate | 0.000 |

Across 200 games, the Neighborhood player wins 144 times (72% win rate) and loses 56 times against the random player, with no ties. On average, Neighborhood scores about 2648 points compared to 2316 for Random, giving a positive average margin of roughly 332 points. This means that, although the neighborhood heuristic is noisy and occasionally loses to random placement, it still provides a meaningful advantage overall: focusing on local clusters of high-value vertices tends to pay off more often than placing tokens blindly. In the "store placement" interpretation, simply targeting locally dense regions of demand already beats a company that chooses store locations with no strategy at all.

Table 4. Performance of Neighborhood player versus Greedy player in the Voronoi game on graphs. Same setup as Table 3, but here the Greedy player always chooses the highest-value free vertex.

| **Metric** | **Value** |
|---|---|
| Games | 200 |
| Neighborhood (A) wins | 75 |
| Greedy (B) wins | 125 |
| Ties | 0 |
| Avg score – Neighborhood (A) | 2394.270 |
| Avg score – Greedy (B) | 2503.640 |
| Avg margin (A − B) | -109.370 |
| Neighborhood win rate | 0.375 |
| Greedy win rate | 0.625 |
| Tie rate | 0.000 |

When matched against the Greedy player, the Neighborhood algorithm wins only 75 out of 200 games (37.5%) and loses 125 (62.5%), with an average margin of about -109 points. On these random graphs, Greedy's strategy of always picking the highest-value free vertex ends up slightly stronger overall than Neighborhood's local-cluster heuristic. This suggests that just summing value within a fixed radius is not always aligned with the final Voronoi partition: Neighborhood sometimes picks vertices that sit near high-value neighbors but lose those neighbors to later placements or to the opponent’s tokens, while Greedy at least guarantees that it claims the single most valuable remaining vertex each turn. In practice, this result shows that a purely local density measure is not reliably better than a simple value-based rule when we ignore who will eventually own each region.

Table 5. Performance of Neighborhood player versus Influence player in the Voronoi game on graphs.
Here, the Neighborhood heuristic is compared to the territory-based Influence player.

| **Metric** | **Value** |
|---|---|
| Games | 200 |
| Neighborhood (A) wins | 0 |
| Influence (B) wins | 200 |
| Ties | 0 |
| Avg score – Neighborhood (A) | 1612.835 |
| Avg score – Influence (B) | 3317.655 |
| Avg margin (A − B) | -1704.820 |
| Neighborhood win rate | 0.000 |
| Influence win rate | 1.000 |
| Tie rate | 0.000 |

Against the Influence player, the Neighborhood strategy loses every single one of the 200 games, with an average score of about 1613 compared to 3318 for Influence and a large negative margin of roughly -1705 points. This is a dramatic gap: the influence-based algorithm consistently captures roughly twice as much total value as the neighborhood heuristic. The behavior here reinforces the main lesson from the exploration section: heuristics that explicitly model ownership changes and marginal territory gain are far more effective than those that only consider local structure or density. Conceptually, Influence is not just asking “where is there a lot of value nearby?” but “where can I actually take value away from the opponent or from unclaimed regions?”, and that shift in perspective translates into a huge performance difference.

Overall, I evaluated VoronoiNeighborhoodPlayer using the same experimental framework as in the main exploration. I added matchups between Neighborhood and each of the other three players (Random, Greedy, and Influence) to my ComparePlayers driver, and ran 200 games for each matchup (100 random graphs * 2 orderings). Against the random player, Neighborhood won 144 out of 200 games (72% win rate) with an average margin of about +332 points, so it is clearly better than random but far from dominant. Against the greedy player, Neighborhood actually lost 75-125 overall, with an average margin of about −109 points, meaning Greedy tends to do slightly better on these graphs. Finally, against the influence-based player, Neighborhood lost all 200 games, trailing by about −1705 points on average. The outcome is that the neighborhood heuristic is middle-of-the-pack; it does beat random in most cases, but it is worse than greedy and much worse than the influence-based strategy.

- **How to Run**

Compile and run the ComparePlayers class under the ext directory. In the main method, I added additional calls to compare of the form compare(VoronoiNeighborhoodPlayer.class, VoronoiRandomPlayer.class, ...), compare(VoronoiNeighborhoodPlayer.class, VoronoiGreedyPlayer.class, ...), and compare(VoronoiNeighborhoodPlayer.class, VoronoiInfluencePlayer.class, ...), using n = 100, density = 0.1, numTurns = 10, and timeLimitMs = 500.