# Analysing Strategies in 2D Voronoi Area Game

Samir Dileep  Sandipan Samanta

National Institute of Science Education and Research, Bhubaneswar

## Problem Statement

This project aims to design and train a Machine Learning model to master the 2-player Voronoi Area Game in 2D. An in-house replica of the game was made in python and the training process was run on this interface. After training, we expect the model to make the most optimal placement of points to win an instance of the game when placing as Red or as Blue. The model also helps answer which of Red and Blue have the advantage when both play optimally.

## Voronoi Diagrams

A Voronoi diagram is a method of geometrical partitioning of a space where the division is based on the distance of every point in the space from a specific set of points. The distance metric used here is Euclidean, but any other metric can be used, depending on the application.

Let $X$ be a metric space with a distance metric $d$. Let $P = \{P_1, P_2, ..., P_n\}$ be the set of points to be considered as sites based on which the division is done. Then, the region $R_i$ assigned to a point $P_i \in P$ will be given by:

$$R_i = \{x \in X \mid d(x, P_i) \leq d(x, P_j) \, \forall \, P_j \in P; \; P_i \neq P_j\} \tag{1}$$

Put simply, the Voronoi diagram can be defined as the collection of $R_i$ from all $P_i$ in $P$.

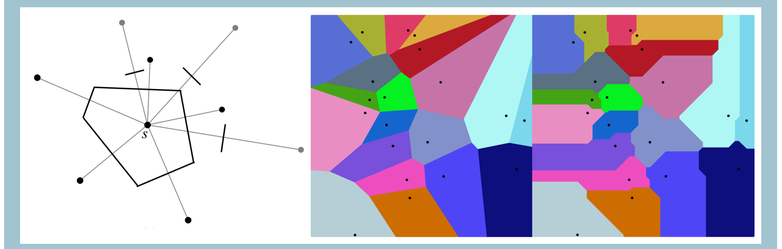
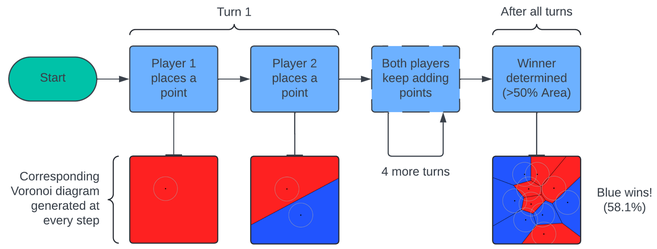



Figure 1. (Left to right) Generation of associated region for a point in a Voronoi diagram; A pair of Voronoi diagrams having the same set of points but a different distance metric in each (Euclidean and Manhattan respectively) [2].

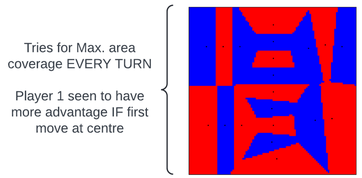
## Applications

This version of the Voronoi game can be used to simulate and find solutions for multiple cases of **Competitive Facility Location** problems, where multiple players place facilities inside a fixed area such that their region of influence is maximised [1]. Changes can be made in no. of dimensions, turns and players to suit each problem. Due to this being a rather simple representation, other required factors like time can be easily incorporated.

## The 2D Voronoi Area Game

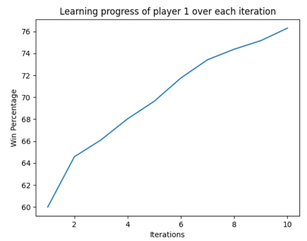


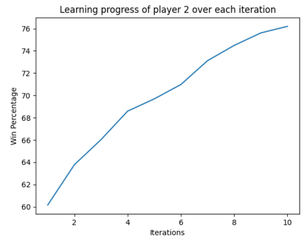

## The Greedy Approach




## The ML Approach

```
Initialize the number of iterations --> num_iterations
Initialize the number of games per iteration--> num_games
Initialize the path to the initial dataset --> data_path

Function run_iterations(num_iterations, num_games, data_path):
    For i from 0 to num_iterations - 1:
        Load dataset from "data_{i}.csv"
        Train KNN model on loaded dataset
        Initialize an empty list --> new_games

        For j from 1 to num_games:
            new_game = simulate_game(model, loaded dataset)
            Append new_game to new_games

        Create a DataFrame from new_games
        Save DataFrame to "data_{i+1}.csv"

Function simulate_game(model, dataset):
    Initialize current_state as an empty list
    Initialize game_data as an empty list

    For t from 1 to 2*size (number of turns):
        If t is even (KNN model's turn):
            If current_state is not empty:
                Predict the next move using KNN model and current_state
            Else:
                Select a random move from the dataset column t
            Append the move to current_state
        Else (Random player's turn):
            Select a random move from the dataset column t
            Append the move to current_state

    Calculate areas for final positions
    Append area results to game_data
    Return game_data

Call run_iterations(num_iterations, num_games, data_path)
```




## Caveats & Further Work

Limitations:

- Model Selection: Initial experiments with PPO, DQN, and A2C were promising at first, but actual implementation over the environment we created turned out to be very difficult and gave little to no results.
- The KNN Model: KNN has given out satisfactory results, providing models for both players 1 and 2 with a win percentage of above 70%. Nevertheless, we feel the previously mentioned algorithms would fare even better, provided we achieve proper implementation.

Performance Issues:

- Greedy Method Drawbacks: The implemented greedy method was too slow to generate multiple games, so some further steps in our approach like pitting greedy VS KNN and the proper statistics for greedy strategy were not obtained.

Future Directions:

- Towards Other Variants: Future efforts will focus on adapting our approach for more complex scenarios like weighted Voronoi games to add more real-life applications.
- Optimizing Computational Efficiency: We plan to enhance the speed and efficiency of our models, particularly improving the greedy method for faster decision-making capabilities. Also, we plan to debug and optimize the codes for PPO, DQN and A2C we wrote earlier in hopes of better performance.

## References

[1] Aritra Banik, Bhaswar B. Bhattacharya, Sandip Das, and Satyaki Mukherjee. The discrete voronoi game in r2. *Computational Geometry*, 63:53–62, 2017.

[2] Francesco S. Bellelli. F.s.bellelli: The fascinating world of voronoi diagrams, 2021.

[3] Bruno Bouzy, Marc Métivier, and Damien Pellier. Mcts experiments on the voronoi game. 2012.

[4] Frederik Brasz. Voronoi diagram area game, 2015.

[5] Wikipedia contributors. Voronoi diagram — Wikipedia, the free encyclopedia, 2024.

## Codes and Resources

For all the codes and resources used in this project, checkout our github page.

samir.dileep@niser.ac.in



sandipan.samanta@niser.ac.in