POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Quality Diversity for Racing Track Design

Tesi di Laurea Magistrale in
Computer Science Engineering - Ingegneria Informatica

Author: **Martino Piaggi**

Student ID: 225662
Advisor: Prof. Pierluca Lanzi
Co-advisors: Prof. Daniele Loiacono
Academic Year: 2024-25

Look out there. Out there is the perfect lap. No mistakes. Every gear change, every corner, perfect. You see it? Most people can't. Most people don't even know it's out there, but it is. It's there.

Ken Miles — Ford v Ferrari

Abstract

Quality Diversity (QD) refers to a class of evolutionary algorithms that focus on finding high-performing yet diverse solutions. They have been applied in several areas, including the exploration of search spaces for video game content. In this thesis, we investigate the use of quality diversity algorithms as tools for the procedural generation of racing tracks for the TORCS [1] and Speed Dreams [2] open-source racing games; the two games use an almost identical representation of the tracks. Specifically, we apply the MAP-Elites algorithm [3] to generate racing tracks for these games, automatically using evaluation functions computed by simulating racing competitions with existing AIs. Our methodology involved developing novel genotype representations based on Voronoi diagrams and convex hulls, an end-to-end pipeline for generation and evaluation, and the use of dimensionality reduction for robust behavioral characterization.

Keywords: Quality Diversity, MAP-Elites, Procedural Content Generation, Racing Tracks, TORCS, Evolutionary Algorithms

Abstract in lingua italiana

La Quality Diversity (QD) denota una classe di algoritmi evolutivi che si concentrano sulla ricerca di soluzioni ad alte prestazioni ma diverse tra loro. Sono stati applicati in diverse aree, inclusa l'esplorazione dello spazio delle soluzioni per i contenuti dei videogiochi. In questa tesi, indaghiamo l'uso di algoritmi di quality diversity come strumenti per la generazione procedurale di tracciati di gara per i giochi di corsa open-source TORCS [1] e Speed Dreams [2]; i due giochi utilizzano una rappresentazione quasi identica dei tracciati. In particolare, applichiamo l'algoritmo MAP-Elites [3] per generare tracciati di gara per questi giochi, utilizzando automaticamente funzioni di valutazione calcolate simulando competizioni di gara con IA esistenti. La nostra metodologia ha comportato lo sviluppo di nuove rappresentazioni genotipiche basate sui diagrammi di Voronoi e sugli inviluppi convessi, una pipeline end-to-end per la generazione e la valutazione, e l'uso della riduzione della dimensionalità per una robusta caratterizzazione comportamentale.

Parole chiave: Quality Diversity, MAP-Elites, Generazione Procedurale di Contenuti, Tracciati di Gara, TORCS, Algoritmi Evolutivi

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

1 Introduction **1**

2 Background and Related Work **3**
2.1 Track design in racing games . . . 3
2.2 Procedural Content Generation (PCG) in racing game development . . . 7
2.3 Literature on Racing Track Generation . . . 10
2.4 Quality Diversity and MAP-Elites . . . 12

3 Methodology **17**
3.1 TORCS . . . 17
3.2 Track Representations . . . 18
3.2.1 Perlin Noise with Polar Coordinates . . . 19
3.2.2 Convex Hull Methods (Maciel Method) . . . 19
3.2.3 Voronoi Diagrams . . . 19
3.2.4 Encoding the Track as a Genotype . . . 20
3.2.5 Variation Operators . . . 21
3.3 MAP-Elites Implementation with Pyribs . . . 26
3.3.1 The Pyribs Conceptual Framework . . . 27
3.3.2 Implementation within the Pyribs Framework . . . 27
3.4 Implementation Details and Pipeline Overview . . . 28
3.4.1 Generation and track visualization . . . 28
3.4.2 Containerization . . . 33
3.4.3 Telemetry and Evaluation . . . 33
3.4.4 Experiment Management and Checkpointing . . . 35
3.4.5 System Architecture and Workflow . . . 36

4 Results and Analysis **39**
4.1 Preliminary Analysis . . . 39
4.1.1 Emergent Features Noisiness Analysis . . . 39
4.1.2 Correlation and Clustering Analysis of Metrics . . . 44

4.1.3 Dimensionality reduction experiments 48
4.2 Experiments 67
4.2.1 Convex Hull Technique Experiments 68
4.2.2 Voronoi Technique Experiments 70

5 Conclusions and Future Developments **81**

Bibliography **85**

A Source Code **91**

Acknowledgements **93**

1 | Introduction

Video game creation demands significant resources for content production. Level design, a central aspect of this process, involves constructing playable environments. Generating sufficient content for engaging experiences and replayability presents a challenge. Procedural content generation (PCG) addresses this by automatically creating digital assets through predefined algorithms, a technique utilized in many games to provide varied gameplay experiences and reduce manual design overhead.

A key challenge in PCG involves generating content that exhibits high performance and broad diversity. Traditional optimization methods often focus on maximizing a single objective, leading to a single best solution. However, diverse player preferences and game contexts require a wider range of high-quality solutions. Quality Diversity (QD) algorithms offer a different approach. They identify numerous high-performing solutions distributed across a behavior space, drawing inspiration from biological evolution. This approach helps explore the search space comprehensively.

This thesis investigates the application of Quality Diversity algorithms to the procedural generation of racing tracks. Specifically, the research applies the Multi-dimensional Archive of Phenotype Elites (MAP-Elites) algorithm to generate tracks for the open-source racing simulators TORCS and Speed Dreams. The system automatically evaluates generated tracks using functions derived from simulated racing competitions with existing AI drivers.

This work develops novel genotype representations based on Voronoi diagrams and convex hulls, an end-to-end pipeline for track generation and evaluation, and incorporates dimensionality reduction for robust behavioral characterization. The research began with preliminary experimentation to quantify the inherent noisiness of emergent gameplay features. This analysis informed the selection of robust behavioral descriptors and the refinement of the fitness function by mitigating unreliable metrics that were susceptible to simulation noise.

Following this, correlation and clustering analyses were conducted to understand metric relationships, and dimensionality reduction techniques were employed for behavioral descriptor generation.

The subsequent chapters detail this research. Chapter 2 presents background on track design in racing games, procedural content generation, and Quality Diversity algorithms. Chapter 3 outlines the methodology, discussing TORCS, track representations, genetic operators, and the MAP-Elites implementation using Pyribs. Chapter 4 provides results from these preliminary analyses and the main experiments, which involved running the

MAP-Elites algorithm to generate tracks and comparing the performance of the different track representations. Chapter 5 concludes the thesis, summarizes key findings, and proposes future research directions.

2 | Background and Related Work

2.1. Track design in racing games

Racing games have undergone significant evolution since their inception, becoming a cornerstone of the gaming industry. A crucial aspect of this evolution lies in the racing tracks, which have transformed from simple layouts to dynamic and immersive environments. The journey began with Space Race (1973) by Atari [4], where players guided rockets in a race against time. Although its mechanics were simplistic and it wasn't strictly a racing game, it introduced the concept of dynamic competition in a digital environment. While track design was non-existent in this title, it set the stage for later games to explore the importance of environment in shaping gameplay. A decade later, Namco raised the bar with Pole Position (1982) [5], a landmark title featuring realistic track layouts and racing mechanics. Players raced on circuits modeled after real-life tracks, striving to master lap times. Pole Position established the template for modern racing games by emphasizing track realism, blending player skill with immersive visuals that made the tracks feel dynamic.

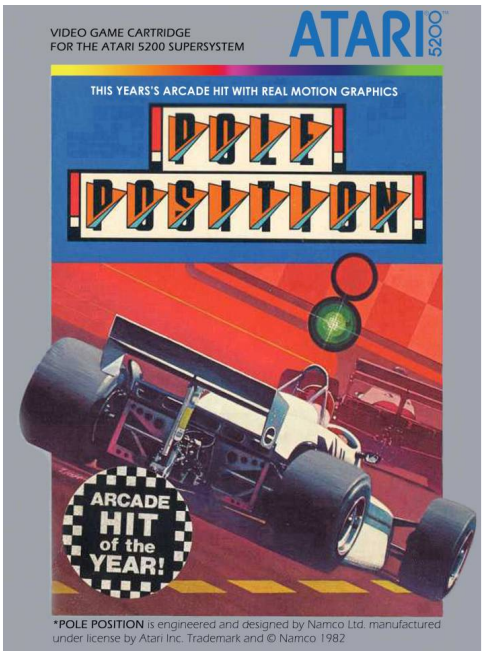

Figure 2.1: Screenshot of Namco's *Pole Position* (1982), a landmark title emphasizing track realism.

In 1986, Sega's Out Run further revolutionized the genre by offering a non-linear driving experience with multiple routes and a choice of background music. This branching track design introduced replayability and personalization, as players could choose their path through the game. The innovation of diverging tracks highlighted how design decisions could elevate player engagement, making tracks an active element in gameplay rather than a static backdrop. [6] The 1990s introduced futuristic racing with Nintendo's F-Zero (1990), utilizing the SNES's Mode 7 graphics to create a sense of 3D racing on a 2D console. The game's track design set a new standard for speed and challenge, featuring winding layouts that required precise control. The high-speed tracks of F-Zero were designed to test players' reflexes, with narrow paths and sharp turns that became defining features of the genre. [7] Two years later, Sega's Virtua Racing (1992) introduced 3D polygonal graphics to the racing genre, providing multiple camera angles and a more immersive experience. This leap to 3D allowed for more complex track designs, incorporating elevation changes and varied layouts that pushed players to adapt their strategies. Virtua Racing laid the groundwork for future 3D racing games to experiment with both realistic and fantastical track environments. [8]

Figure 2.2: Screenshot from Sega's *Virtua Racing* (1992), showcasing early 3D polygonal graphics in racing games.

Across these early franchises, a common design challenge emerged: the delicate balance between realism and player accessibility. Since then, each franchise has introduced its unique vision of what a racing experience should be. Super Mario Kart, introduced in 1992 [9], transformed the notion of track design into a playground of creativity. The tracks featured fantastical elements, such as shifting terrains, environmental hazards, and interactive obstacles such as thwomps and banana peels. These elements introduced unpredictability and tactical decision-making, turning the tracks into dynamic arenas of engagement rather than precise technical courses. Shortcuts, hidden paths, and power-ups added depth, allowing players to develop strategies beyond pure speed. Mario Kart's approach redefined the genre, introducing accessibility and chaos that broadened the appeal of racing games, particularly for social gatherings and younger players.

In 1997, Polyphony Digital released Gran Turismo, a game that set new standards for realism in racing simulations. [10] The series has continued to evolve, with Gran Turismo 7 releasing on March 4, 2022, for PlayStation 4 and PlayStation 5 [11]. Developers employ advanced techniques such as laser scanning to replicate every incline, camber, and surface variation, making tracks not just backdrops but dynamic environments that challenge players' technical skills. The mastery of the track in Gran Turismo requires precision in braking points, throttle control, and cornering, reflecting the depth that detailed track design brings to the genre. Similarly, Assetto Corsa, developed by Kunos Simulazioni, has been acclaimed for its realistic driving simulation and focus on detailed track recreation. [12] The original game was released on December 19, 2014, and its successor, Assetto Corsa EVO, entered Early Access on January 16, 2025 [13]. The series emphasizes authentic car handling and utilizes laser-scanned tracks to achieve unparalleled realism. Each track demands technical precision from the players, as variations in elevation, camber, and cornering complexity provide a challenging and rewarding experience. This attention to detail makes Assetto Corsa a standout example of how track design can elevate the simulation genre. Since 2015, the Milestone s.r.l.-developed The *Ride* series is also notable for its authentic motorcycle handling and extensive customization options. [14] Tracks in *Ride* are designed to highlight the dynamics of motorcycle racing, offering players challenges that emphasize tight cornering, elevation changes, and the interplay of speed and control. By bringing motorcycles into the simulation genre, *Ride* expands the diversity of track design within racing games.

These simulation-focused titles demonstrate a clear design philosophy: the track is a technical puzzle to be solved. Quality is therefore measurable through geometric properties that create challenge—complex corner combinations, variations in camber that affect grip, and sequences that demand precise rhythm from the driver.

Figure 2.3: Screenshot from The *Ride* series, illustrating its focus on realistic motorcycle racing.

In contrast, the Forza Horizon series, developed by Playground Games, offers a more arcade-like experience with open-world environments. [15] The first installment was released on October 23, 2012, and the latest, Forza Horizon 5, launched on November 9, 2021 [16]. Set in a fictionalized representation of Mexico, Forza Horizon 5 features diverse landscapes, including rocky mountain paths, lush forests, and arid deserts. Tracks in Forza Horizon invite exploration and reward players for embracing risk-taking and experimentation. The open-world format encourages players to explore unconventional routes and off-road adventures, creating a sense of freedom rarely found in structured racing games. Dynamic weather further enhances the challenge, as players must adapt to conditions like rain-slicked roads or fog-obscured paths. This flexibility and variety make track design in Forza Horizon a key component of its appeal.

Further expanding the concept of track diversity is the rise of user-generated content, exemplified by the *TrackMania* series [17]. By providing players with intuitive yet powerful track editors, these games outsource content creation to the community, resulting in a virtually limitless repository of circuits ranging from simple loops to complex, physics-defying challenges. This paradigm underscores the value of exploring a vast design space, a principle central to procedural generation.

Customization of vehicles also emerges as a crucial aspect of racing game design, closely tied to track demands. Need for Speed, Forza, and Ride stand out, offering players opportunities to tailor their vehicles for specific track conditions. Adjusting parameters like tire grip, suspension stiffness, and aerodynamic profile allows players to align their performance with the unique demands of each track. This level of customization deepens the player's connection to the game, as vehicles become extensions of their strategies.

Tracks, as a core element of racing games, intertwine technical and creative dimensions. Fundamental elements such as race lines and clipping points delineate optimal paths while variations in track width and camber offer overtaking opportunities. Height variations add complexity and can be used by artists to increase the visual appeal of the gameplay. Visual cues such as road markings, environmental details, and distinctive landmarks ensure players remain oriented during high-speed races. Track design, with all these elements, shapes the immediate racing experience.

2.2. Procedural Content Generation (PCG) in racing game development

Procedural Content Generation (PCG) is the automated creation of digital assets using predefined algorithms that require minimal user input. This approach allows for the dynamic generation of diverse and engaging content, significantly enhancing the scalability and variability of game design [18]. Notable examples of PCG in the video game industry include *Minecraft*, where PCG generates vast, block-based landscapes filled with biomes, caves, and intricate underground networks. Similarly, games like *Spelunky* and *Diablo III* leverage PCG to enrich replayability by randomizing caves, dungeons, and loot systems. *The Binding of Isaac*, another acclaimed game, incorporates PCG in its dungeon design, offering randomly generated rooms, enemies, and items with every playthrough. This randomness makes each session challenging and unpredictable, requiring players to adapt their strategies constantly. *No Man's Sky* stands out for its groundbreaking use of PCG to create a procedurally generated universe, featuring over 18 quintillion unique planets, each with its own ecosystems, terrains, flora, and fauna. This scale guarantees players will encounter fresh discoveries, fostering a sense of endless adventure.

Togelius, Yannakakis, Stanley, and Browne [19] propose a comprehensive taxonomy of PCG methodologies, delineating several approaches:

1. **Online vs. Offline Approaches**: Online methods dynamically generate content during gameplay, responding to player actions in real time. Offline methods, by contrast, pre-generate content during the development phase, ensuring stability and thorough testing before deployment. Online PCG emphasizes real-time efficiency and adaptability, while offline PCG prioritizes quality assurance and pre-emptive debugging.

2. **Stochastic vs. Deterministic Methods**: Stochastic methods leverage probabilistic algorithms to introduce variability and unpredictability, enhancing replayability. Deterministic methods are based on predefined rules that ensure consistent and predictable outcomes. For example, stochastic generation can create diverse terrains and environments, while deterministic methods ensure critical elements, e.g., pathways or quest objectives, remain intact and balanced.

3. **Constructive vs. Generate-and-Test Methods**: Constructive methods focus on generating content that is immediately usable and adheres to quality standards without requiring additional refinement. In contrast, generate-and-test methods iteratively produce, evaluate, and optimize content to achieve desired outcomes,

often requiring more computational resources but yielding highly tailored results. Generate-and-test methods employ fitness functions to evaluate and refine content iteratively, ensuring alignment with predefined quality metrics.

The fitness function of generate-and-test methods can be direct (fast to compute and well-suited for representation with few features), simulation-based (testing the content in a simulated environment), or evaluated by humans through game data or questionnaires. Simulation-based fitness functions are especially useful in games with complex interactions, as they can simulate and evaluate player behaviors or AI interactions to refine content. Procedural techniques can also be categorized as assisted and non-assisted methods. Assisted techniques require significant user intervention, offering precise control and customization, while non-assisted methods generate content autonomously, suitable for large-scale environments or real-time generation.

Referring to the taxonomy of PCG approaches just defined, Search-Based Procedural Content Generation (SBPCG) represents a specialized subset of generate-and-test PCG. SBPCG utilizes search algorithms, such as evolutionary computation, to explore a vast space of possible game assets. By iteratively generating, evaluating, and refining content against specific criteria, SBPCG ensures that assets meet predefined goals for quality, balance, and player engagement. As seen in terrain and city generation, iterative methods such as genetic algorithms play a critical role in generating optimized content through repeated cycles of evaluation and refinement. Additionally, SBPCG can employ multi-objective optimization to balance competing goals, such as difficulty, aesthetics, and playability.

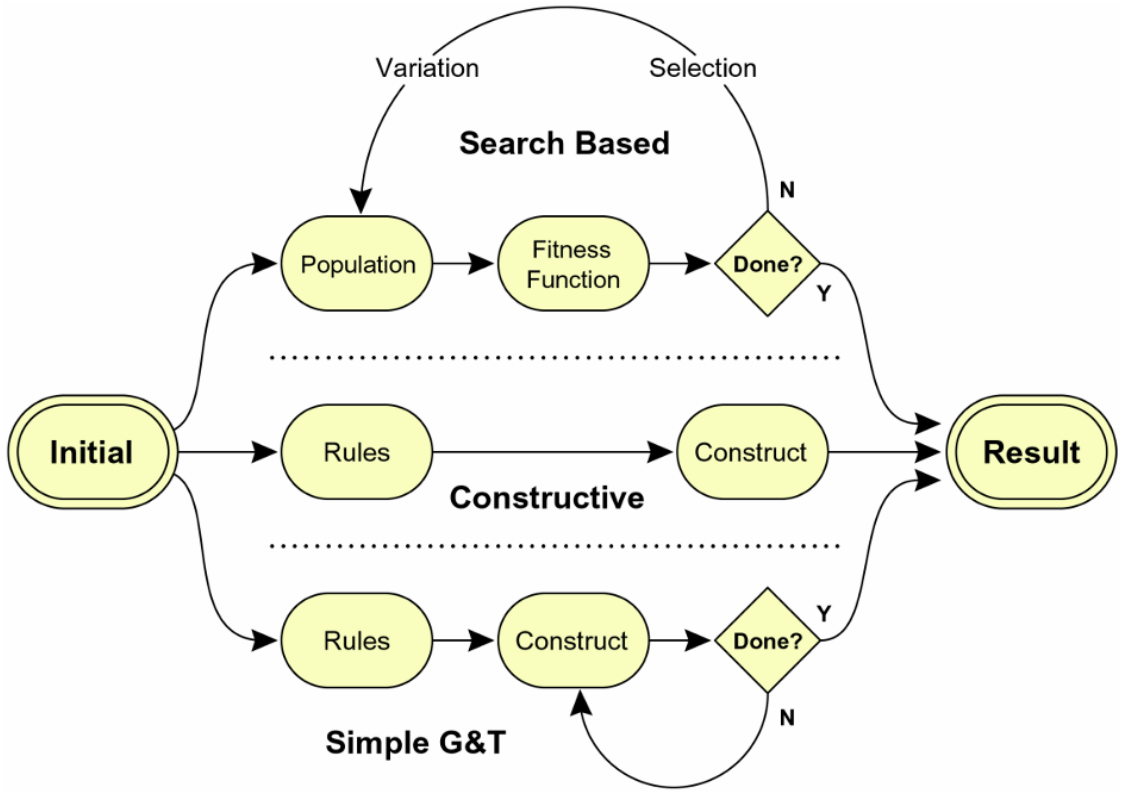

Figure 2.4: Conceptual diagram of the Search-Based Procedural Content Generation (SBPCG) Loop.

An extension of SBPCG is Evolutionary Algorithms (EAs), which are inspired by the principles of natural selection. The solutions are optimized following a framework where the concepts of genotype and phenotype are fundamental to the outcome of the search. Genotypes are low-level representations that the algorithm manipulates, while phenotypes represent the actual solutions evaluated by the fitness function. Effective genotype design is essential to maintain locality, ensuring that small changes in the genotype result in proportionally small changes in the phenotype. This balance facilitates efficient exploration of the solution space. Direct encoding maps a genotype to a phenotype, simplifying implementation but often resulting in larger search spaces. Indirect encoding, on the other hand, employs compact representations such as random seeds to reduce search space complexity. While this approach is more efficient, it may sacrifice some granularity in solution refinement. For instance, indirect encoding is effective for generating large terrains or intricate mazes, while maintaining computational efficiency.

Notably, Togelius et al. [20, 21] and Loiacono et al. [22] evolved tracks optimized for factors such as challenge levels and driving styles, using evolutionary algorithms. These methods involve defining fitness functions that guide the evolutionary process toward desirable track features. Fitness functions play a crucial role in guiding evolutionary algorithms, serving as criteria to evaluate and select content for survival and reproduction. In

racing track generation, fitness functions may measure quantifiable aspects such as track length, curvature, and difficulty. For example, Togelius et al. [20] used fitness functions to balance playability and challenge in evolving tracks. Simulation-based approaches, on the other hand, evaluate tracks through gameplay simulations, often using AI-controlled agents. Interactive fitness functions go a step further by incorporating human feedback, as seen in Cardamone et al. [23]. This approach leverages human intuition and subjective preferences to guide the evolution of game content, ensuring alignment with player expectations. This human input helps refine the evolutionary process, resulting in tracks that are not only technically sound but also enjoyable to play.

In the context of racing games, PCG can generate track layouts, vehicles, or environments. This thesis focuses specifically on racing tracks, which are central to the genre's identity.

2.3. Literature on Racing Track Generation

The review of existing literature on racing track generation reveals a diverse array of techniques and methodologies employed to achieve both realism and playability. Building on the prior section, this analysis incorporates relevant insights and methodologies to provide further elaboration on procedural content generation (PCG) within the specific context of racing games.

Procedural content generation methods have shown immense versatility, with road network generation emerging as a particularly intricate challenge. Drawing from techniques employed in virtual city creation, spline-based road generation offers a robust approach. The application of cubic and Bézier splines, as detailed in [20, 22], allows for the creation of smooth, continuous road geometries. These splines ensure that generated roads maintain a balance between aesthetic quality and navigability. For racing track applications, spline-based techniques can be augmented with procedural textures and UV mapping, which are critical for generating realistic and visually coherent road meshes [24].

Beyond direct spline manipulation, other procedural methods leverage fundamental geometric constructs to ensure desirable track properties, such as being a closed loop. An example of this is the convex hull approach, detailed by Maciel [25]. This technique begins with a set of randomly generated 2D points. The monotone chain algorithm is then used to compute the convex hull that encloses these points, yielding a foundational, albeit angular, closed path. To transform this into a functional racetrack, subsequent smoothing algorithms, such as spline interpolation, are applied to create a continuous and drivable circuit. The simplicity and reliability of this method are notable, as it provides a deterministic way to generate a wide variety of track shapes simply by altering the initial distribution of points.

The use of Voronoi diagrams as a spatial partitioning mechanism offers another layer of sophistication in environment modeling [26]. Voronoi diagrams are fundamental constructs in computational geometry that partition a plane into regions based on the proximity to a set of seed points, often referred to as "sites." These regions, known as Voronoi cells, are convex polygons that collectively form a tessellation of the plane. Each cell corresponds to a site and contains all points closer to that site than to any other.

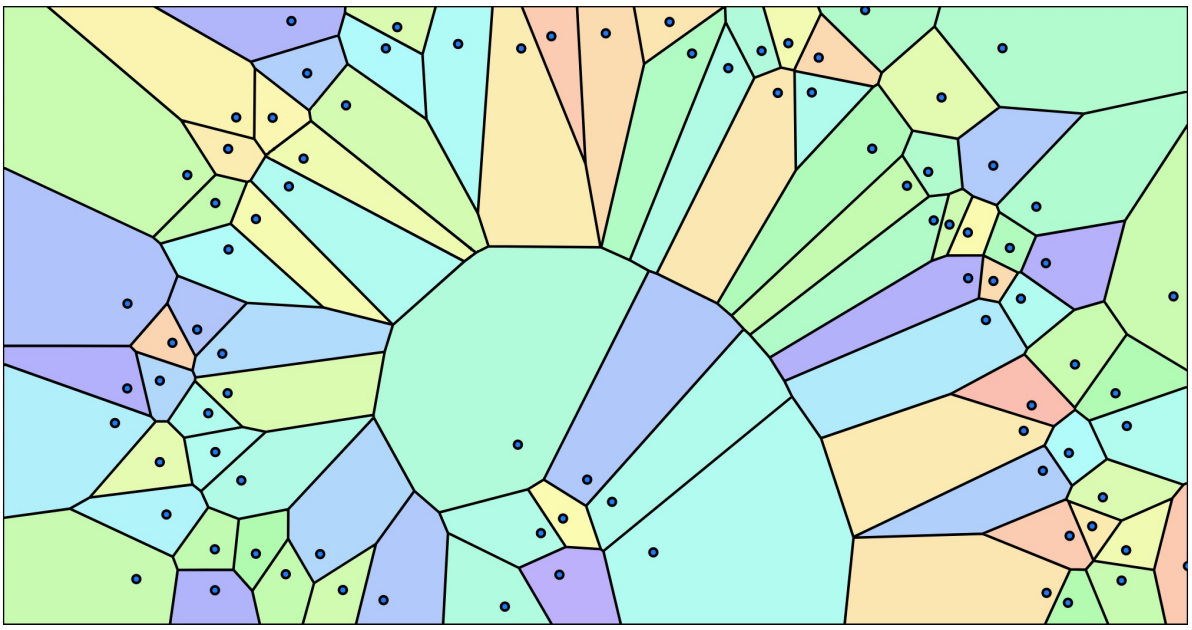

Figure 2.5: An example of a Voronoi diagram, where the plane is partitioned into cells based on proximity to a set of seed points (sites).

While these properties have been extensively applied in procedural city generation to create realistic road networks and subdivide urban layouts [27–29], their direct application to racing track generation is less common, representing a more novel area of investigation. A key and highly relevant example is the Camilla procedural-track engine [30], which specifically uses Voronoi diagrams to generate urban-style racing environments. In this engine, adjusting the density and distribution of the Voronoi seed points directly yields tracks with varying complexities and street weave patterns, demonstrating the technique's direct potential for the racing genre. This approach highlights how the abstract geometric properties of Voronoi diagrams—where edges represent points equidistant from two sites—can be effectively harnessed to create diverse and structured track layouts.

To turn such a conceptual network into a fully functional and visually coherent asset, further techniques are required. For instance, the generated Voronoi edges can be converted into smooth spline paths and then textured using procedural UV-mapping to create realistic road meshes [24]. Furthermore, should the network contain intersections, advanced detection algorithms can be employed to prevent graphical issues like overlapping meshes and texture flicker at junctions [31]. These supporting methods, often developed in the context of broader procedural world generation [32], are crucial for translating the high-level Voronoi structure into a polished in-game track.

Another noteworthy innovation is the hybridization of procedural methods with backtracking algorithms. Freiknecht [24] introduces an intersection modeling technique wherein splines are combined with a backtracking approach to ensure that intersections occupy minimal space while retaining navigational coherence. This method emphasizes the translation of procedural patterns into flat, mesh-based geometries—optimizing performance

without compromising visual fidelity. Such techniques hold significant promise for runtime applications in racing games, especially when paired with shader-based texturing for real-time rendering efficiency.

Constraint satisfaction problem (CSP)-based methods offer additional utility in segmenting racing tracks into modular components. By defining parameters for curvature, length, and width, CSP-based approaches facilitate the assembly of tracks that align with predefined constraints, ensuring both structural integrity and gameplay diversity. Wang and Missura [21] effectively demonstrate the application of CSP in creating dynamic and engaging track layouts by segmenting tracks into fixed-length straight lines, left turns, and right turns.

The integration of machine learning further expands the horizon of PCG in racing track design. Generative Adversarial Networks (GANs) have been explored for their ability to generate racetracks that mimic real-world layouts. Tanzola [33] exemplifies the use of GANs to generate 2D racing tracks by training on datasets of existing tracks. While computationally intensive, this data-driven approach captures nuanced design patterns, enabling the creation of tracks with high visual and structural fidelity.

Despite the advancements in procedural generation, challenges remain in balancing computational efficiency and quality. Real-time generation, in particular, demands algorithms that can dynamically create coherent, playable tracks without significant computational overhead. In conclusion, the literature on racing track generation highlights the convergence of traditional geometric methods, advanced procedural techniques, and emerging machine learning models. From spline-based designs to GANs, the methodologies emphasize adaptability, scalability, and creativity. Future research should continue to explore the integration of procedural techniques with real-time systems, enabling the generation of immersive, engaging racing environments that align with evolving gameplay expectations.

2.4. Quality Diversity and MAP-Elites

While classical optimization approaches often focus on maximizing a single objective, such as player enjoyment or lap time consistency, Quality Diversity (QD) algorithms offer an alternative by seeking a wide range of high-performing solutions across different regions of the behavior space. This approach aligns with the inherent complexity and variability found in natural systems, where evolution produces diverse species adapted to distinct niches [34].

In the context of racing track generation for video games, procedural content generation (PCG) has often relied on single-objective optimization, such as minimizing track length variance or ensuring balanced curvature. These methods typically identify a single best solution based on a fitness function. However, in practice, "best" is context-dependent. Short, high-intensity circuits may appeal to some players, whereas long, technical tracks might challenge more experienced racers. A singular track design is insufficient to meet diverse player preferences and skill levels. QD algorithms address this limitation by recognizing that many types of "high-quality" solutions can coexist. For example, a short track with frequent overtaking opportunities can provide high competitiveness, while a

long, scenic circuit with sweeping corners may offer a different but equally valuable play experience.

QD algorithms systematically explore the behavior space while preserving quality for each member of the population. Lehman and Stanley pioneered this approach with Novelty Search, a method that promotes behavioral diversity without a specific goal [35]. Building on this, they introduced Novelty Search with Local Competition (NS-LC), which employs a localized fitness pressure to encourage competition among behavioral neighbors [36]. This marked the first true QD algorithm, as the localized fitness pressure facilitated the discovery of diverse, high-quality solutions across behavioral niches.

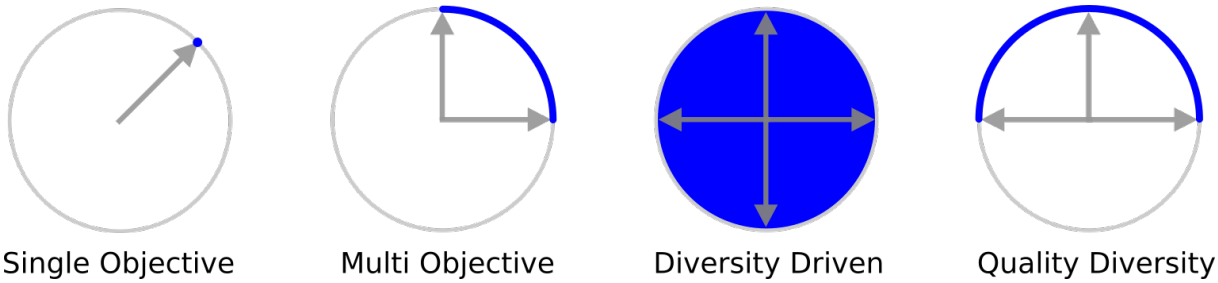

Figure 2.6: Diversity contrasting traditional optimization (focused on a single peak) with Quality Diversity (exploring multiple high-performing regions).

The Multi-dimensional Archive of Phenotype Elites (MAP-Elites) [3] introduces an innovative approach to exploring behavior spaces by partitioning them into discrete cells, each defined by user-specified behavior characterizations. These cells represent distinct niches within the space, where the goal is to populate each niche with the highest-performing solution. MAP-Elites iteratively evaluates candidate solutions based on their behavioral features and performance, ensuring that only the best-performing individual is retained in each cell. This process not only identifies optimal solutions but also provides a comprehensive map of performance across the entire behavior space, enabling insights into trade-offs and relationships between features and fitness. By systematically illuminating the behavior space, MAP-Elites transforms traditional optimization into a tool for discovering diversity and high-quality across multiple dimensions.

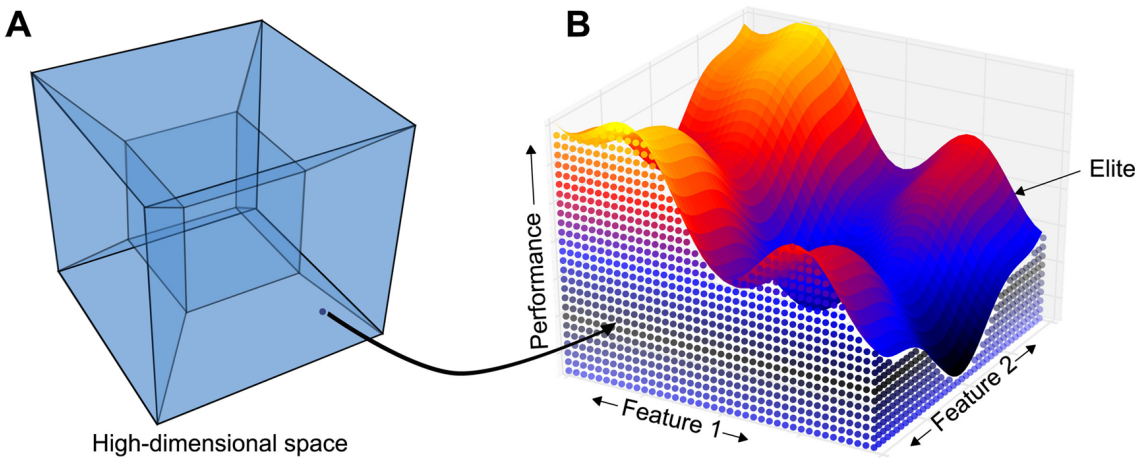

Figure 2.7: Conceptual representation of a multi-dimensional behavior space, illustrating how MAP-Elites partitions the space into discrete cells to explore diverse solutions.

Variants of MAP-Elites have since addressed challenges such as computational scalability and exploration efficiency. For instance, Centroidal Voronoi Tessellation MAP-Elites (CVT-ME) [37] reduces the number of niches in high-dimensional spaces by partitioning the feature space into a fixed number of homogeneous regions, enhancing efficiency without compromising diversity. Covariance Matrix Adaptation MAP-Elites (CMA-ME) [38] combines MAP-Elites with Covariance Matrix Adaptation Evolution Strategy (CMA-ES), improving exploration capabilities through adaptive sampling. Multi-Emitter MAP-Elites (ME-MAP-Elites) [39] employs heterogeneous emitters with distinct strategies to improve search efficiency, while MAP-Elites Low-Spread (ME-LS) [40] focuses on reducing performance variability among elites, ensuring more consistent solutions.

Behavior characterization (BC) plays a pivotal role in the success of QD algorithms. BCs serve as vectors defining behavioral traits of solutions within the search space. Aligning BCs with the ultimate quality metrics helps simplify exploration by improving the efficiency of MAP-Elites. Multi-BC approaches, which utilize multiple behavioral descriptors simultaneously, have been proposed to mitigate limitations in single-BC setups and address the complexity of diverse domains [41]. Furthermore, variants such as MAP-Elites + Passive Genetic Diversity (MEPGD) address the limitations of strict elitism by preserving multiple individuals per bin, facilitating lineage exploration [34]. Similarly, MAP-Elites + Novelty (MENOV) integrates novelty scores into selection, enhancing the exploration of underrepresented behaviors [34].

Descriptor-Conditioned Gradients (DCG) extend MAP-Elites by integrating gradient-based search, which enhances the efficiency and reliability of exploration in high-dimensional behavior spaces. The introduction of a descriptor-conditioned critic in this method ensures that new solutions are evaluated not only for their fitness but also for their contribution to the diversity and quality of the archive. This refinement enhances the ability of QD

methods to address deceptive landscapes effectively and populate challenging regions of the search space [42].

Moreover, Faldor et al. describe how DCG-MAP-Elites incorporates an archive distillation process to consolidate knowledge into a single descriptor-conditioned policy. This policy facilitates interpolation across behaviors without requiring the storage of numerous solutions, significantly reducing computational overhead while maintaining versatility. The approach has demonstrated superior QD-scores and coverage in domains requiring broad exploration, further emphasizing its potential for application in complex environments [42].

In conclusion, QD algorithms, particularly MAP-Elites, with its numerous variants, represent a powerful tool for PCG. Their ability to explore behavior space extensively while retaining high-performing solutions aligns with the goals of enhancing player experience and game replayability. While computational demands and selecting behavioral descriptors pose challenges, these algorithms remain promising for diverse content generation in racing games. Integrating MAP-Elites into the PCG pipeline can create tracks that meet high standards of playability and challenge while offering a rich variety of racing experiences.

3 | Methodology

3.1. TORCS

This study utilizes The Open Racing Car Simulator (TORCS) [1], an open-source, three-dimensional racing platform widely used in both entertainment and academic research. Selected for its modularity, portability, and robust support for AI agent development, TORCS provides a suitable environment for procedurally generating and evaluating racing tracks.

Figure 3.1: A gameplay screenshot from The Open Racing Car Simulator (TORCS).

First developed in 1997, TORCS has become a foundational platform for AI research, notably featured in competitions at the IEEE Conference on Computational Intelligence and Games. Its architecture allows researchers to develop custom controllers and complex driving agents via a low-level API that grants access to the simulation state. The platform's core strengths for this research include its detailed physics engine, support for custom tracks and vehicles, and a non-graphical mode that significantly accelerates simulation times for large-scale experiments. TORCS simulates comprehensive vehicle dynamics—including tire grip, aerodynamics, and suspension—using a discrete-time simulation with a 0.002s step. This combination of features provides a robust and extensible platform for the procedural generation and analysis of racing tracks, enabling detailed investigation into the relationship between track design and emergent driver behavior.

Its extensibility enables the addition of custom tracks and vehicles, and permits core modifications. Additionally, its non-graphical mode increases simulation speeds significantly. This characteristic makes it particularly useful for large-scale experiments and iterative testing. Moreover, TORCS allows for the use of multiple simulated cars, or "robots," each of which has the opportunity to gather and process track information. This permits the calculation of a suitable initial racing line, car setup, and team/pit strategy.

In summary, TORCS provides a suitable framework for this research due to several key characteristics. Its open-source and extensible architecture was leveraged to integrate the custom telemetry modules required for our analysis. Moreover, its capacity for non-graphical execution is well-suited to the large-scale, automated evaluations inherent in the MAP-Elites algorithm. Collectively, these features create a practical environment for studying the relationship between procedurally generated track layouts and the emergent dynamics of the simulator's AI agents.

Moreover, several forks of TORCS exist. One of them, Speed Dreams, is geared towards a better human player experience. Speed Dreams is a racing simulation game that, while sharing its origins with TORCS, has evolved to emphasize a more refined user interface and experience, providing higher quality graphics and enhanced gameplay. Speed Dreams is often used for human gameplay, while TORCS is favored for research. Other forks include pyTORCS, which is a port of TORCS to Python, replacing several modules with standard open-source software.

3.2. Track Representations

This section details the methods employed to represent racing tracks in this study. In procedural content generation (PCG) for racing games, the processes of encoding and decoding are crucial to transform abstract representations into tangible game elements. Specifically, genotype-to-phenotype mapping refers to the transformation of a compact, encoded representation (the genotype) into a usable game asset (the phenotype). For racing track generation within the TORCS simulator, this involves encoding track characteristics into a set of parameters and subsequently converting these parameters into a track layout that TORCS can interpret and render. This section delineates the methodologies used to encode racing tracks using Voronoi diagrams, the mapping of these encoded representations into TORCS-compatible formats, and the strategies implemented to ensure the reliability and diversity of the generated tracks.

The representation in PCG significantly influences the diversity, complexity and playability of the generated content. This section examines three primary methods: Perlin noise with polar coordinates, convex hull methods (specifically the Maciel method), and Voronoi diagrams. The discussion assesses each method's suitability for generating diverse and engaging tracks. Smoothing algorithms, such as spline interpolation, and validation mechanisms are applied to ensure the generated tracks are both aesthetically appealing and drivable within TORCS.

TORCS represents tracks internally as a circular, doubly linked list of segments. Each segment includes information about its curvature, width, surface type, and barriers. This

structured representation is essential to achieving the research goals of this thesis, as it allows for a direct translation of the generated track into the simulation.

3.2.1. Perlin Noise with Polar Coordinates

Initially, Perlin noise with polar coordinates was considered for its capacity to generate smooth, closed-loop tracks with inherent variability. Its straightforward implementation made it an attractive starting point. However, experiments revealed that Perlin noise, while capable of producing aesthetically pleasing tracks, did not provide the desired level of diversity and control. While increasing the number of noise layers and adjusting parameters such as frequency and amplitude could create a range of track designs, from flat, fast circuits to complex and twisty layouts, the method ultimately had limited control over specific track features and resulted in repetitive designs. For instance, fine-tuning elements like curve placement or segment length proved difficult, impacting the balance between playability and track difficulty. This method also required meticulous parameter tuning and the layering of multiple noise functions to achieve sufficient variability. In summary, Perlin noise with polar coordinates, although useful for creating smooth, continuous, and closed paths, lacked the precise control and diversity required for this project.

3.2.2. Convex Hull Methods (Maciel Method)

To overcome the limited control and diversity offered by Perlin noise, we adopted a method based on convex hulls, adapting the approach pioneered by Maciel [25]. In our implementation, the genotype is defined by a set of 2D points. From these points, we compute the convex hull using the efficient monotone chain algorithm. The resulting polygonal boundary is then smoothed with spline interpolation to form a drivable track. This technique offers two key advantages for our research: first, its deterministic nature provides a reliable foundation for generating topologically valid, closed-loop tracks. Second, it offers an intuitive mechanism for exploring the design space, as altering the distribution and density of the initial points directly influences the track's shape and complexity with minimal parameter tuning.

3.2.3. Voronoi Diagrams

Building on the potential of Voronoi diagrams for procedural generation, as outlined in Chapter 2, we selected this method as a primary representation for our track genotypes due to its geometric expressiveness. In our approach, a track's underlying structure is derived from the edges of a Voronoi tessellation generated from a set of seed points.

To build a continuous racing track, a random distribution of seed points is used to generate Voronoi cells. The edges of these cells form a graph network that defines the underlying track structure. Algorithms then process the graph to extract a continuous path by tracing the edges of selected cells. To enhance the drivability and visual aesthetics of the track, smoothing techniques like spline interpolation using Catmull-Rom splines or Bézier curves are applied. These methods remove sharp corners and ensure smoother transitions

between track sections.

The advantages of Voronoi diagrams in this context are numerous. Their ability to inherently produce closed-loop paths ensures that tracks can be seamlessly generated without requiring additional closure logic. Furthermore, the flexibility in manipulating seed point distributions permits the exploration of diverse track geometries, spanning simple layouts to intricate patterns. The computational efficiency of algorithms for constructing Voronoi diagrams—such as divide-and-conquer, plane sweep, and higher-dimensional embedding techniques—has further cemented their importance in computational geometry.

Voronoi diagrams can efficiently tackle complex design and optimization problems, such as generating diverse, visually appealing, and functional track layouts.

3.2.4. Encoding the Track as a Genotype

A racing track's final representation in TORCS requires adherence to a specific XML schema that defines key elements such as straights, curves, surfaces, and barriers. In practice, this detailed layout—the phenotype—is what the racing engine parses and renders during simulation. The engine demands that the track is precisely defined in terms of its segment lengths, angles, and other physical attributes.

The design process begins with a compact and abstract encoding of the track's structure, designated as the genotype. As discussed in the previous section, in our research the genotype is constructed using two complementary strategies: one based on Voronoi diagrams and the other on convex hull methods. These approaches capture the core geometric and topological features of a track while reducing the dimensionality of the design space, which simplifies subsequent genetic operations.

The Voronoi-based genotype is derived from a collection of seed points that are randomly distributed within the target domain. Each seed point, acting as a site, contributes to a Voronoi cell. A subset of these cells is carefully selected so that their edges form the continuous path of the track layout. The genotype therefore consists of the coordinates of the original seed points along with the coordinates of the sites of the selected Voronoi cells. This selection is critical because the outline of the track is determined by the edges of these cells.

In the convex hull approach, the genotype comprises simply the coordinates of the original seed points. The convex hull is computed using a deterministic algorithm—in our implementation, the monotone chain algorithm—to produce a unique boundary that encloses the set of points. The resulting trajectory is then refined through smoothing techniques to enhance drivability. This deterministic mapping from seed points to convex hull eliminates randomness in the final layout, providing a reliable basis for generating the racing track.

Using these compact, low-dimensional genotypes allows for the efficient application of crossover and mutation operators during the evolutionary search.

Once a promising genotype is evolved through an evolutionary operation, it undergoes a mapping process, and a solution compatible with TORCS track format is created.

During mapping, the abstract geometric constructs are transformed into a network of track segments, smoothed using algorithms e.g., spline interpolation to guarantee a drivable path, and finally formatted into TORCS's native XML format. The conversion process is delicate; even minor errors in segment continuity or curvature can result in a track that fails to close properly or that violates gameplay requirements.

In summary, representing a racing track in TORCS's XML format is the final goal of a two-stage process. First, an abstract genotype, encoded through either Voronoi diagrams or convex hull methods, encapsulates the track's fundamental structure. Next, a precise mapping process translates that abstract representation into a fully specified, simulation-ready phenotype. This separation between genotype and phenotype streamlines both the design and optimization phases conforming to the technical constraints of TORCS engine.

3.2.5. Variation Operators

Building on the foundations of the genotype discussed in the previous section, this chapter focuses on the crossover and mutation operators developed during the research. These genetic operators play a critical role in exploring and exploiting the design space, enabling the generation of diverse and high-quality racing tracks. This section outlines the challenges faced during the implementation of these operators, details the specific strategies adopted for mutation and crossover, and evaluates their impact on the resulting tracks.

Designing effective crossover and mutation operators for racing track generation presents several significant challenges. A primary concern is maintaining the structural integrity of the track, ensuring that any offspring remains a well-formed and playable circuit. Genetic operations must preserve this integrity, avoiding not only discontinuities that break the closed-loop structure but also other geometric defects like self-intersections or abrupt transitions that could compromise playability. Another challenge lies in achieving a balance between exploration and exploitation within the design space. Operators must facilitate the discovery of novel track configurations while retaining desirable features from existing designs. This balance ensures that the genetic properties of high-quality tracks are preserved while allowing for innovation.

Crossover operators combine segments from two parent tracks to create new offspring, ensuring the resulting tracks are continuous and playable with smooth transitions at segment junctions. Strategies include merging parent track sections or recombining control points to maintain coherence, directly influencing track diversity and quality by determining how genetic material is shared across generations. Mutation operators introduce controlled variations by adjusting parameters like control point positions or segment curvature. These changes must balance preserving most of the track structure mutating only for example a single seed point position.

Effective mutation and crossover operators are crucial for avoiding local optima and ensuring the evolutionary process explores the design space efficiently. Computational efficiency is also vital, as operators must rapidly generate and evaluate tracks to support the iterative nature of evolutionary algorithms.

Implementation of Mutation Operators Two main mutation strategies are used: one for Voronoi-based genotypes and another for convex hull-based genotypes. For Voronoi-based genotypes, the process randomly selects a site from the set of chosen cells that define the track. The selected site's x and y coordinates are each altered by a small, randomly generated displacement, with the magnitude of these changes governed by an intensity parameter. This adjustment produces minor geometric variations in the track while preserving its continuity. The convex hull-based mutation operates on the points that form the boundary of the track. In this approach, one selects a random point from the convex hull representation and applies small, intensity-scaled displacements to its coordinates. The corresponding point in the complete seed point dataset is then updated to reflect this change. This method specifically targets the track's outer limits, inducing localized geometric variations that can subtly affect the overall track shape without compromising the closed-loop quality inherent in the convex hull. For both mutation operators, the intensity hyperparameter can be tuned to balance exploration and preservation. Low-intensity mutations create slight adjustments that fine-tune a track's layout, while higher intensities promote broader diversity. In practice, these mutation operators complement crossover methods by introducing localized changes after genetic information has been recombined. They help ensure that the evolutionary process explores the design space preserving the validity of each solution, for example without allowing the number of selected cells to grow uncontrollably. This controlled variability is crucial for generating tracks that are both structurally coherent and visually representative of successful design patterns inherited from parent genotypes. Overall, these mutation strategies are simple in design yet effective; they introduce targeted modifications that maintain the essential geometric and topological characteristics of the tracks.

Implementation of Crossover Operators The following section presents a detailed description of the crossover operators used in this research. The two distinct genotypic representations (Voronoi diagrams and convex hulls) required specialized crossover strategies tailored to their unique data structures. This section elaborates on the methods applied, the challenges encountered, and the rationale for the chosen implementations.

Crossover Strategy for Voronoi-Based Genotypes Designing an effective crossover operator for Voronoi-based genotypes is a non-trivial task. The operator must intelligently combine the geometric and topological features of two parent tracks to produce viable, coherent offspring. A successful crossover is fundamental to the evolutionary algorithm's search capability, as it must balance the exploration of novel track configurations with the exploitation of successful traits inherited from the parents. A poorly designed operator might consistently produce invalid tracks or fail to create meaningful new structures, stalling the evolutionary process. To address this challenge, several distinct crossover strategies were developed and evaluated, each with a different approach to merging the parental genetic material.

During development, the following strategies were implemented and considered:

1. **Random-Line Partitioning Method:** The implementation begins by calculating the geometric center of the selected Voronoi seed points from both parent tracks.

This center serves as a reference for generating a random dividing line. The line's slope is determined by a random angle selected from a uniform distribution, while its intercept is computed to ensure the line passes through the center. Once the dividing line is established, each parent's Voronoi cells are partitioned into two subsets depending on their relative position to the line. Two possible divisions are evaluated: one that assigns cells on one side of the line to the offspring and another that reverses this assignment. The division that maximizes the balance between the number of cells from the two parents is selected. The offspring's final genotype is the union of the selected cells, ensuring adequate representation from both parents.

2. **Relative Reconstruction Method:** This approach attempted to overlay one parent's selected Voronoi cells onto the other by aligning their geometric centers. Cells from one parent were spatially shifted relative to the center of the other parent. However, this method introduced instability because shifting cells caused boundary distortions, often compromising the offspring's structural integrity.

3. **Midpoint Method:** This method calculated the midpoints between corresponding seed points from two parents and used these midpoints as the sites for the offspring. While geometrically sound, this approach was ultimately discarded after initial tests. It consistently produced offspring that deviated significantly from both parents, resulting in overly simplistic or unrecognizable track layouts that lacked the desired complexity.

4. **Connection-Based Method:** This heuristic-based method was implemented to repair cases where Voronoi-based crossovers produced disconnected track segments. A breadth-first search (BFS) algorithm was applied to identify and add bridging cells, ensuring connectivity. Although this method improved continuity, it was more effective as a supplement rather than as a standalone approach.

A critical component of all strategies is a regularization stage to address potential "explosions" in the number of cells selected. Offspring with too many Voronoi cells can become computationally infeasible and geometrically unmanageable. To prevent this, a target cell count, typically the average number of cells from the two parents, is maintained. Cells farthest from the offspring's center are iteratively removed until the target count is achieved. This trade-off preserves the structural core of the offspring while eliminating redundant or excessive elements.

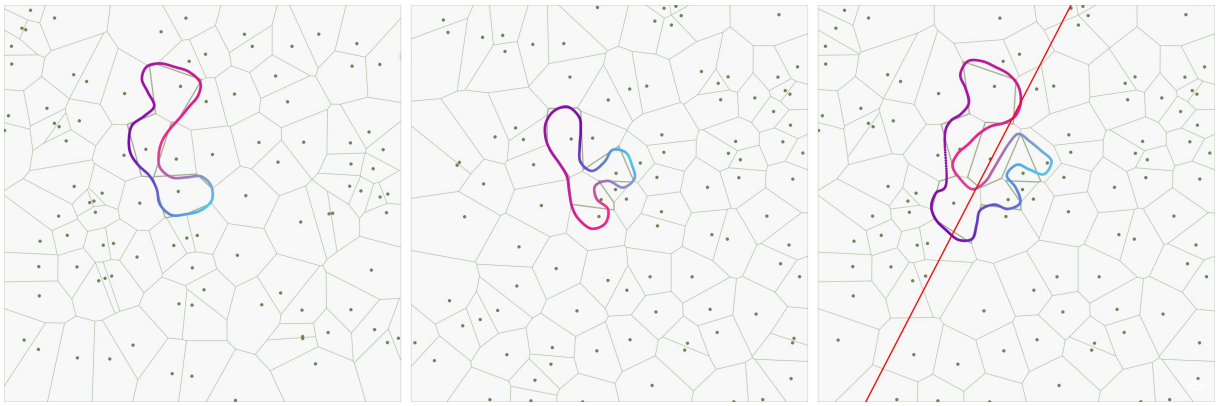

Figure 3.2: Illustration of the Random-Line Partitioning Method for Voronoi-based crossover. Parent tracks are divided by a random line, merging selected cells to form the offspring. The connection-based method supplements cases with distant cells.

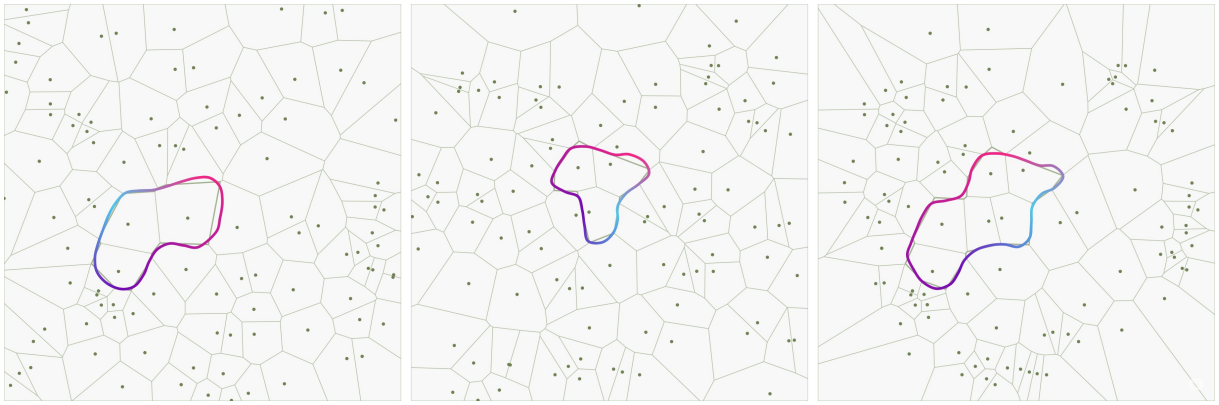

Figure 3.3: Illustration of the Relative Reconstruction Method for Voronoi-based crossover. The Voronoi diagrams of two parent tracks are combined to produce the offspring shown on the right. Cells from one parent are spatially shifted relative to the center of the other, along with their neighboring sites, to preserve the overall cell shapes and structure during merging.

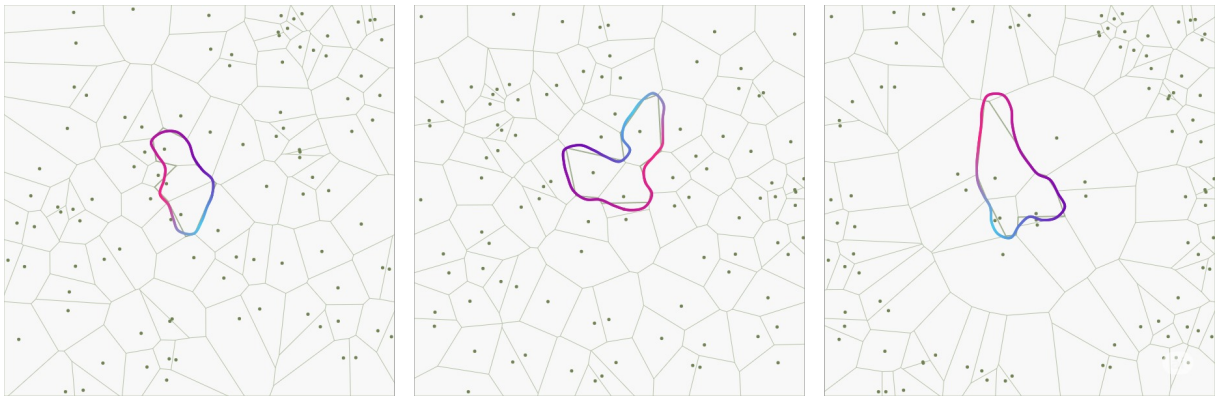

Figure 3.4: Illustration of the Midpoint Method for Voronoi-based crossover. The Voronoi diagrams of two parent tracks are combined by calculating midpoints between corresponding seed points. It often results in offspring with overly simplistic or unrecognizable track layouts, limiting its effectiveness.

From a purely visual comparison, the *Random-Line Partitioning Method*—augmented by regularization and supplemented by the BFS heuristic when needed—seems to offer the most effective balance between parental trait integration and geometric coherence. However, this is purely visual assessment not supported by quantitative analysis. In the experimental phase of this research, we conducted comparison runs between the *Random-Line Partitioning Method* and the *Convex Hull Crossover Method* to evaluate their performance in terms of track diversity, playability, and computational efficiency. The results of these experiments provide insights into the effectiveness of each crossover strategy in generating high-quality racing tracks.

Crossover Strategy for Convex-Hull-Based Genotypes The crossover operator for convex-hull-based genotypes partitions the point sets of two parents using a dividing line. This line is defined by a randomly generated slope and is calculated to pass through the geometric center of the combined parent points.

Each parent's point set is then split based on its position relative to this line. One subset of points is taken from the first parent (e.g., all points below the line), and a complementary subset is taken from the second parent (e.g., all points above the line). These two subsets are merged to form the initial offspring genotype.

A critical post-processing step enforces a fixed size on the offspring's point set, ensuring it matches the parents' count. If the merged set is too large, it is randomly shuffled and truncated. If it is too small, points are randomly sampled with replacement from the merged set until the target size is reached. This explicit size control, rather than an inherent property of the method, prevents uncontrolled growth in the number of points. The final genotype is later smoothed with spline interpolation to create a drivable track.

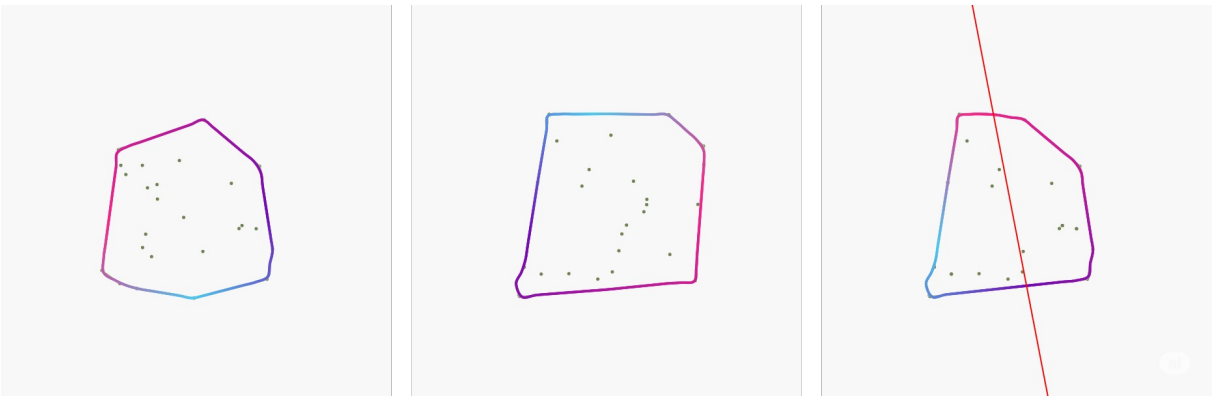

Figure 3.5: Illustration of the Convex Hull Crossover Method. The left shows the convex hulls of two parent tracks, divided by a randomly oriented line passing through their combined center. The right displays the offspring track layout formed by merging subsets of points from both parents.

Conclusion In the context of the entire framework, the crossover operators are designed to balance diversity and feasibility for both Voronoi and convex hull genotypes. Offspring must retain sufficient resemblance to their parents while introducing novel features that expand the design space. Throughout the development process, multiple variants of these crossover operators were implemented and considered, including those detailed in the preceding subsections.

A key observation during development was that each approach benefited from visual inspection to confirm the offspring's structural validity and visual appeal. Tracks that deviated too far from parental features or exhibited disconnected segments were flagged and revised, leading to iterative refinements in the operators' design. These qualitative evaluations were critical in ensuring the developed operators could reliably generate coherent and varied track structures. The subsequent experimental analysis (as detailed in Chapter 4) quantifies the performance of these different crossover strategies, including the Random-Line Partitioning Method for Voronoi cells and the ordinary least squares regression-based approach for convex hulls, to understand their respective contributions to the evolutionary search.

3.3. MAP-Elites Implementation with Pyribs

This section details the practical implementation of the Multi-dimensional Archive of Phenotype Elites (MAP-Elites) algorithm, which serves as the core of our evolutionary search. We utilized the Pyribs library [43], a modern Python framework designed for Quality Diversity (QD) optimization. Pyribs was chosen for its modular architecture, flexibility, and robust implementation of state-of-the-art QD algorithms, making it an ideal platform for our research.

3.3.1. The Pyribs Conceptual Framework

Pyribs implements a modular conceptual framework called RIBS, which unifies methods like MAP-Elites and its many variations. This framework is built on three main components that work in concert:

- **The Archive**: This is the central data structure that stores the collection of elite solutions. It is structured as a grid (or "map") partitioned by the behavioral dimensions of the problem. When a new solution is submitted, the Archive determines its corresponding niche and adds it only if the niche is empty or if the new solution has a higher fitness score than the existing elite.
- **The Emitters**: Emitters are responsible for generating new candidate solutions to be evaluated. Their complexity can range from a simple emitter that randomly selects an elite from the archive and applies mutation, to more advanced strategies like the CMA-ME emitter, which uses the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to intelligently guide the search.
- **The Scheduler**: The Scheduler is the high-level orchestrator that coordinates the interaction between the Archive and the Emitters. It follows an "ask-tell" interface: it "asks" the emitters to generate a new batch of solutions. After these solutions have been evaluated externally by the user's own system, the Scheduler is "told" the results, which it then passes to the emitters and the archive to update their respective states.

This workflow is inherently modular. A key design principle of Pyribs is that it deliberately does not handle solution evaluation, leaving this complex, domain-specific task entirely to the user. Our pipeline leverages this by integrating Pyribs with our external simulation and analysis tools.

3.3.2. Implementation within the Pyribs Framework

Our implementation translates the abstract components of Pyribs into a concrete system tailored for procedural track generation.

Archive Configuration Given that the potential range of track characteristics was not known in advance, we employed Pyribs' `SlidingBoundariesArchive`. Unlike a static grid, this advanced archive type dynamically adjusts its boundaries based on the solutions it observes, which is more suitable for exploratory search. The archive was configured to remap its boundaries periodically and to maintain a buffer of the most recent solutions, allowing it to adapt to the evolving population.

Custom Emitter for External Operators To integrate our bespoke genetic operators, we implemented a `CustomEmitter` that inherits from Pyribs' 'EmitterBase'. This class is the bridge between the Python-based evolutionary loop and our Node.js-based geometry engine. It operates in two phases:

1. **Initialization:** To bootstrap the search, the emitter first generates an initial popu-

lation by making API calls to our external generation server, creating entirely new, random track genotypes.

2. **Evolution:** After initialization, the emitter drives the search by selecting parents from the archive and applying either crossover or mutation. These genetic operations are not performed in Python; instead, the emitter makes API calls to our backend server, which executes the complex geometric manipulations.

This design effectively decouples the high-level evolutionary orchestration from the low-level geometric logic.

Defining the Search Problem for Pyribs The abstract problem of track design is made concrete for Pyribs through the following representations:

- **Genotype:** Each solution is represented internally as a one-dimensional NumPy array—a flattened, fixed-length vector encoding the track's seed points. Helper functions handle the conversion between this array format and the structured JSON required by our external services.
- **Behavioral Vector and Fitness Score:** The evaluation of each track is handled by the broader pipeline infrastructure. For each solution, this pipeline returns the two key pieces of information required by the Pyribs scheduler: a **behavioral vector** (the solution's coordinates in the diversity map) and a single **objective score** (its fitness).

When the 'tell' method of the scheduler is called with these two pieces of data, it updates the archive, completing the evolutionary cycle. This allows Pyribs to orchestrate the search based on the quality and diversity information provided by our external evaluation pipeline.

3.4. Implementation Details and Pipeline Overview

This chapter presents the technical framework that integrates the generation, simulation, and evaluation of procedural racing tracks. The pipeline orchestrates several components: Python supports the MAP-Elites algorithm and data analysis, Node.js drives track generation and real-time visualization using Processing.js, and Docker manages TORCS simulations. A custom HTTP API, implemented in Node.js with Express, connects track generation, simulation, and telemetry data.

3.4.1. Generation and track visualization

A crucial requirement for this research was the ability to visually inspect and debug the geometric outputs of our procedural algorithms. To address this need, we developed a web-based visualization tool using p5.js [44], a modern JavaScript library for creative coding. Its capability for direct, canvas-based rendering and interaction made it ideal for visualizing the construction of tracks and for the iterative debugging of our genetic operators.

The selection of a JavaScript library for the frontend directly informed our decision to use Node.js for the backend. This choice created a cohesive, full-stack JavaScript environment, which streamlined development and ensured seamless communication between the client-side visualizer and the server-side generation logic. This integrated setup proved highly practical, providing the immediate visual feedback necessary to validate and refine our geometric algorithms efficiently.

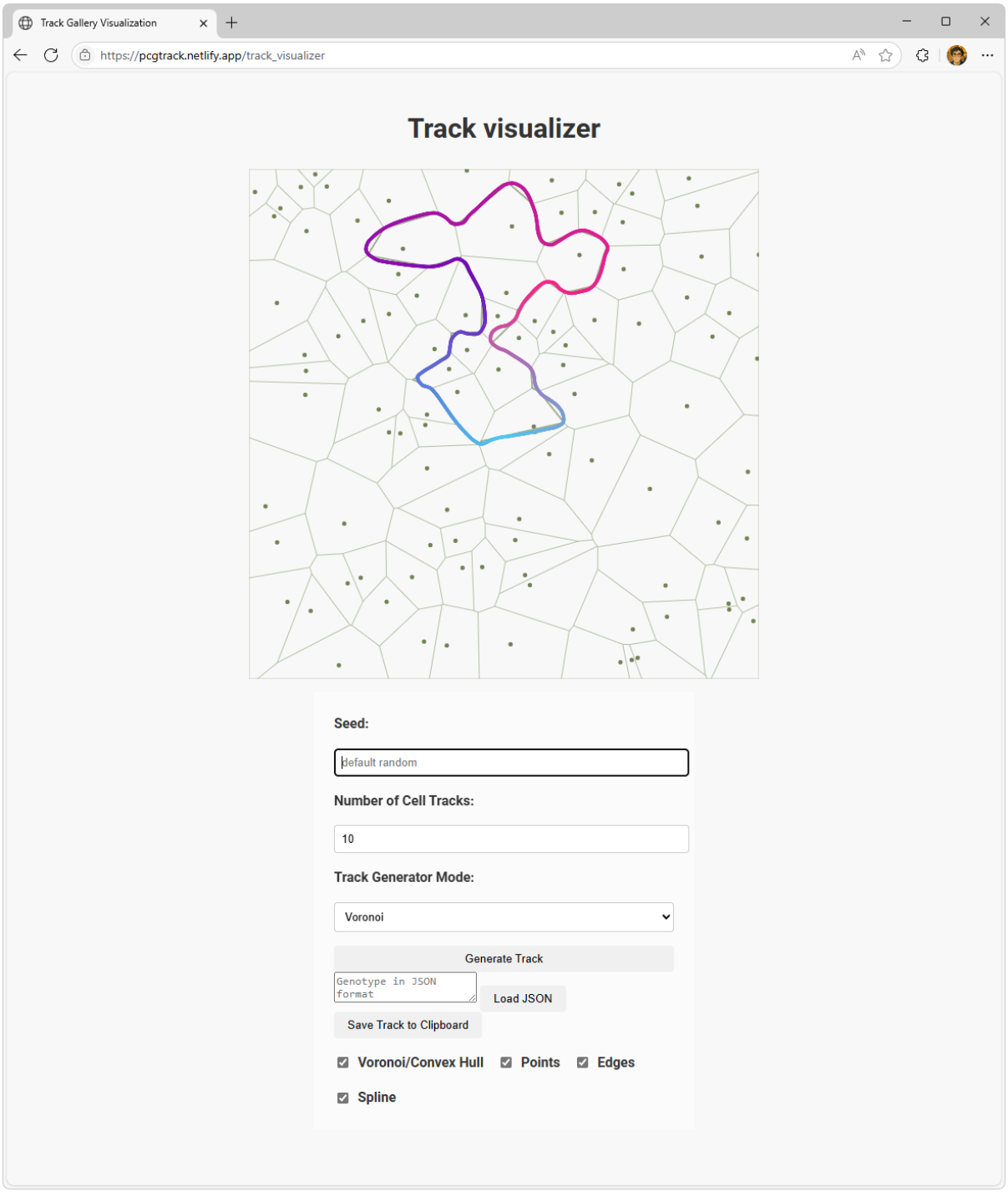

Figure 3.6: Screenshot of the web-based track visualizer, developed using p5.js, enabling real-time track generation and parameter adjustments.

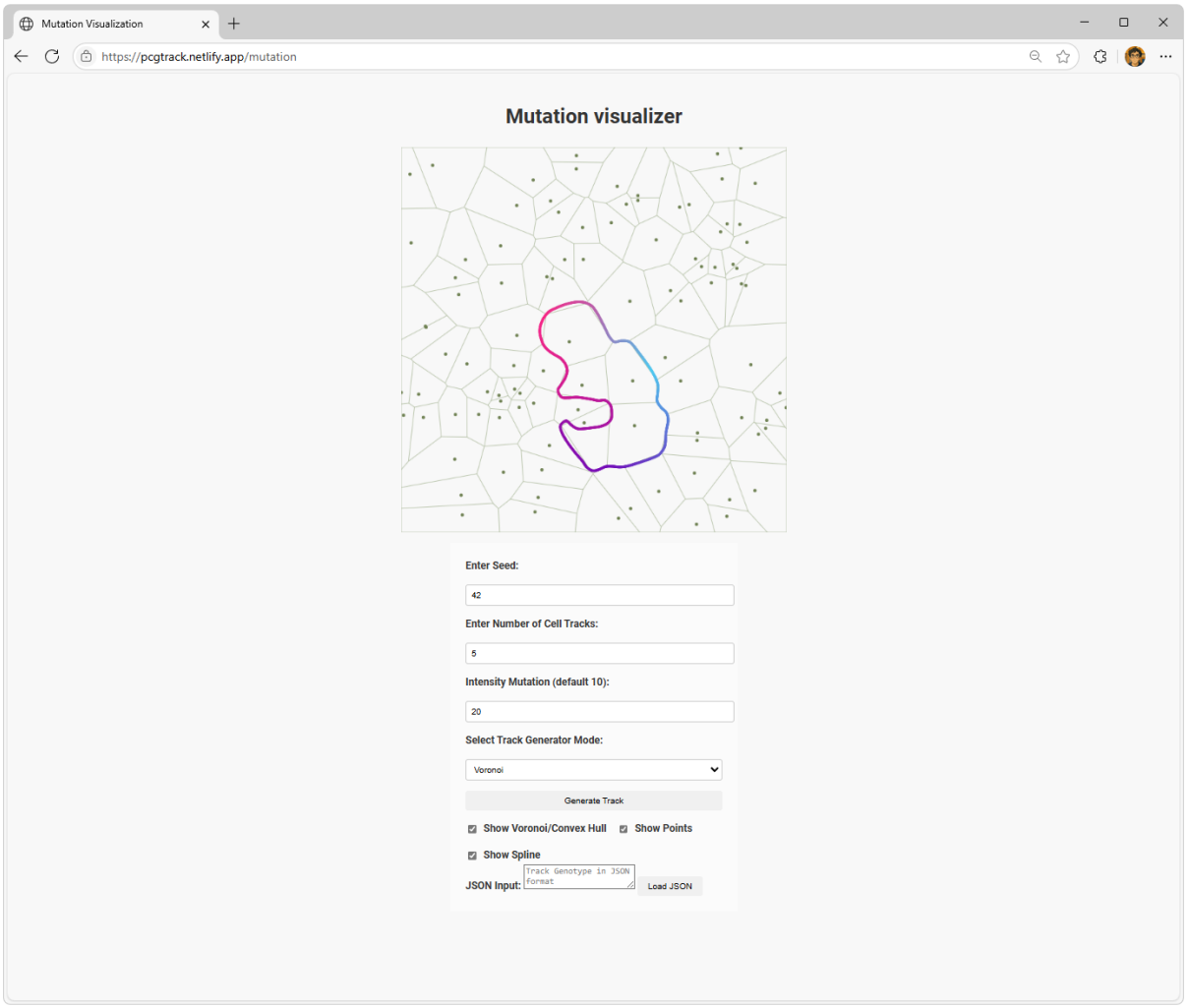

Figure 3.7: Screenshot of the web-based mutation operator visualizer.

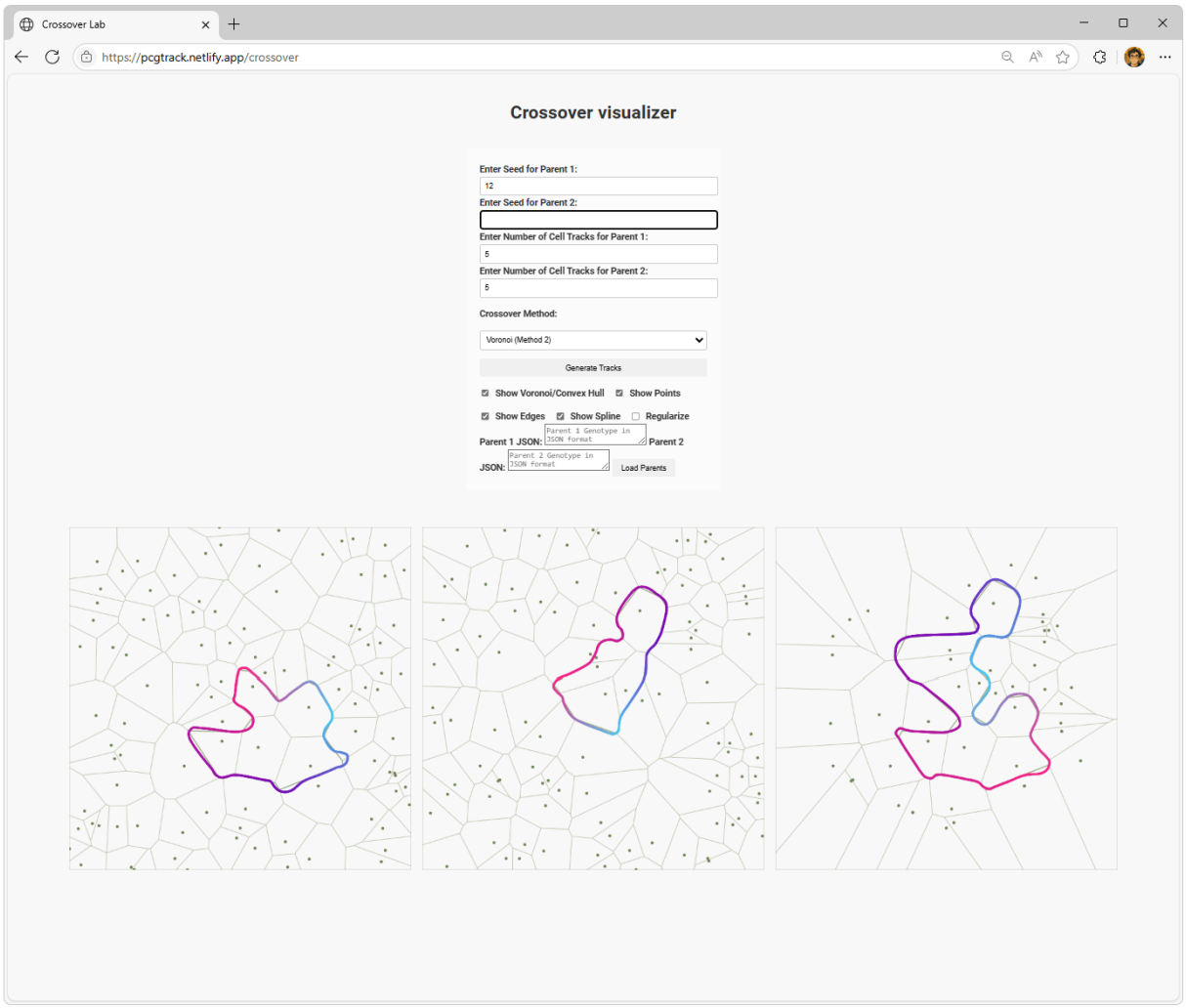

Figure 3.8: The p5.js-based interface demonstrating the Relative Reconstruction crossover method. Structural features from the two parent tracks (left) are merged to produce a viable offspring (right) that inherits traits from both.

As shown in the website's screenshots, the p5.js-powered interface enabled real-time visualization of the generated tracks and the application of genetic operators. All track generation research phases were visualized and analyzed, from Voronoi diagrams to convex hulls, points, edges, and splines. Users can input seeds, select generator modes, and adjust parameters in real time. The visual interface also supports loading, saving, and comparing JSON representations of tracks. These JSON representations were a useful aid, created to facilitate a manual evaluation of the solutions found during the MAP-Elites iterations, allowing for quick visualization and cross-checking of the results.

Following the interactive phase, the Node.js [45] backend converts abstract genotypes, represented as Voronoi or convex hull parameters, into functional tracks compatible with the TORCS simulator. It uses JavaScript libraries to process the track geometry and incorporates error handling mechanisms that validate and correct formatting issues, preventing invalid tracks from reaching the simulation stage.

Moving to the rest of the pipeline and its integration, the Node.js environment hosts a custom HTTP API built with the Express framework. This API manages communication between the main Python environment and the backend services. The evolutionary algorithm itself was managed within a Jupyter Notebook [46]. A Jupyter Notebook is a web-based interactive computing environment that allows for the combination of live code, visualizations, and narrative text in a single document. This makes it an ideal tool for exploratory research, allowing for the direct execution of the MAP-Elites loop, real-time monitoring of the archive's progress, and immediate analysis of the results. The Notebook communicates with the API to coordinate the entire pipeline, from track generation to the parallelized simulation in Docker containers.

3.4.2. Containerization

A significant technical challenge in this research was managing the TORCS simulation environment. TORCS has a complex set of legacy dependencies, making manual installation prone to inconsistencies that could corrupt experimental results. To solve this, we employed Docker [47] for containerization, creating a portable, self-contained, and reproducible simulation environment. Our 'Dockerfile' precisely defines this environment, starting from an Ubuntu base and installing all dependencies, including the X Virtual Framebuffer (Xvfb) for headless operation and our custom telemetry modules [48].

Instead of maintaining a persistent pool of containers, our Node.js server orchestrates this architecture using a single-use container strategy. A fresh container is instantiated for each track evaluation and is destroyed immediately afterward. This approach provides resilience; should a simulation freeze or crash, the faulty container can be safely timed out and shut down without affecting the main server or other concurrent evaluations.

3.4.3. Telemetry and Evaluation

The TORCS simulation, containerized with Docker, collects data using custom telemetry tools. These tools are built upon modifications to the TORCS C++ engine by Sirianni [48] that capture raw gameplay metrics during each simulation. The analysis pipeline involves two main stages: first, a Python script within the container orchestrates the simulation runs and aggregates the raw telemetry into a structured JSON output. This output is then captured and parsed by the main Node.js backend, which performs the final feature assembly. Significant custom logic was implemented at this stage to process and format the collected data, ensuring the final telemetry outputs are fully compatible with the Python-based MAP-Elites implementation.

Data collected during simulations allows the derivation of a set of descriptive features. These features characterize both the geometric properties of the generated tracks and the emergent gameplay dynamics they facilitate. They serve as the foundation for both the behavioral characterization dimensions within MAP-Elites and the components of the fitness function.

Geometric Features

These features describe the intrinsic structural properties of the generated racing tracks.

- **Track Length (*length*):** This metric quantifies the total length of the track in meters, calculated as the sum of all individual segment lengths. It provides a fundamental measure of track scale.
- **Average Radius Mean and Variance (*avg_radius_mean*, *avg_radius_var*):** *avg_radius_mean* represents the average radius of curvature across all curved segments of the track. *avg_radius_var* quantifies the spread in these radii, indicating the diversity of curve tightness.
- **Bend Counts (*left_bends*, *right_bends*):** These values represent the total number of left-turning and right-turning segments on the track. These metrics are normalized by dividing by the total track length, providing a ratio that characterizes the track's curvature density independently of its scale.
- **Straight Sections (*straight_sections*):** This value indicates the total count of straight segments on the track. Similarly to bend counts, this metric is normalized by track length to obtain the proportion of non-curving sections relative to the overall track size, making it a more effective descriptive feature that is independent of track scale.

Emergent Gameplay Features

These features are derived from the behavior of AI agents during simulated races, reflecting the dynamic qualities of the track. Several of these metrics leverage **Shannon entropy** to quantify the variability and complexity of different aspects of gameplay. To calculate the entropy for a given continuous feature (e.g., vehicle speed), the set of all recorded values is first discretized into a histogram, using a fixed number of 30 bins in our implementation. The probability p_i for each bin i is then calculated as the fraction of values that fall into that bin. The entropy H is given by the formula:

$$H = -\sum_{i=1}^{N} p_i \log_2(p_i) \tag{3.1}$$

where N is the total number of bins and p_i is the probability of a value falling into the i-th bin. A higher entropy value indicates that the measured quantity is more evenly distributed across its range, signifying greater complexity or unpredictability.

- **Speed Entropy (*speed_entropy*):** This is the Shannon entropy of the distribution of vehicle speeds recorded across all track segments. A higher value suggests a wider range of speeds experienced, indicating a track that demands varied driving dynamics.
- **Acceleration Entropy (*acceleration_entropy*):** This metric quantifies the Shannon entropy of the distribution of vehicle accelerations throughout the race. It measures the variability in acceleration demands on the track.

- **Braking Entropy (*braking_entropy*):** This is the Shannon entropy of the distribution of vehicle braking forces. It quantifies the variability in braking intensity required by the track layout.
- **Curvature Entropy (*curvature_entropy*):** This metric represents the Shannon entropy of the distribution of track segment curvatures (inverse of radius). A higher value indicates a greater diversity of turns and straights, suggesting a geometrically varied track.
- **Total Overtakes (*total_overtakes*):** This value indicates the cumulative number of overtaking events recorded during the simulated race. This raw count provides an initial measure of competitive dynamics.
- **Gap Mean and Variance (*gaps_mean*, *gaps_var*):** These metrics quantify the time differences (gaps) between successive cars at the race's conclusion. *gaps_mean* is the average time gap, and *gaps_var* is its variance. Smaller values indicate closer, more competitive races. The system considers gaps up to a maximum realistic time (e.g., 120 seconds).
- **Positions Mean and Variance (*positions_mean*, *positions_var*):** These statistics describe the distribution of changes in driver positions throughout a race. *positions_mean* indicates the average shift in position, while *positions_var* quantifies the spread of these changes. They reflect the fluidity of competitive positions.

Track Closure Features

These features assess the topological integrity of the generated tracks.

- **Positional Discrepancies (*deltaX*, *deltaY*, *deltaAngleDegrees*):** These metrics quantify the geometric error in track closure. *deltaX* and *deltaY* represent the positional discrepancy (in meters) between the track's start and end points in global Cartesian coordinates. *deltaAngleDegrees* measures the angular difference (in degrees) between the track's initial and final heading. Significant values indicate a track that does not perfectly close, potentially impacting simulation stability and gameplay.

The data for these features is collected and processed into JSON format. This format transmits fitness scores and auxiliary evaluation data to the Python backend. This information is critical for updating the MAP-Elites archive and guiding the evolutionary search toward generating tracks that exhibit high quality and diversity.

3.4.4. Experiment Management and Checkpointing

The iterative nature of evolutionary algorithms, particularly those involving extensive simulations like MAP-Elites, often demands significant computational resources and prolonged execution times. To address the practical challenges associated with long-running experiments, such as unexpected system interruptions, resource limitations, or the need for intermediate analysis and debugging, a robust checkpointing mechanism was integrated into the pipeline.

This mechanism ensures the persistence of the algorithm's state at regular intervals, allowing experiments to be safely paused and subsequently resumed from their last saved point. The entire state of the MAP-Elites *scheduler* and its associated *archive*—encompassing the current population of elites, their behavioral characteristics, and fitness scores—is periodically serialized. This serialization is achieved using Python's *pickle* module, which efficiently converts complex Python objects into a byte stream that can be written to disk. Checkpoints are saved every 50 iterations, creating timestamped backup files (e.g., *checkpoint_ 0050.pkl*, *checkpoint_ 0100.pkl*).

Upon initiation, the system first checks for the presence of existing checkpoint files. If found, the most recent checkpoint is loaded, restoring the *scheduler* and *archive* to their precise state at the time of the last save. This resume capability prevents the loss of valuable computational progress in the event of system failures or interruptions, while enabling experiments to be segmented across multiple sessions or computational environments. Additionally, checkpoints facilitate iterative refinement by allowing researchers to inspect the evolving archive at specific states, analyze emergent track designs, and make informed decisions regarding hyperparameter adjustments without restarting the entire evolutionary process.

This proactive approach to experiment management significantly enhances the reliability, efficiency, and flexibility of the procedural content generation pipeline, transforming what would otherwise be a brittle, high-risk computational endeavor into a manageable and robust research process.

3.4.5. System Architecture and Workflow

The preceding sections have detailed the individual components of our methodology, from the track representations and genetic operators to the containerized simulation environment. This final section provides a holistic overview of how these components are integrated into the complete procedural content generation pipeline, as illustrated in Figure 3.9.

The entire process is orchestrated from an interactive Jupyter Notebook environment, which hosts the Pyribs-driven MAP-Elites engine. The notebook serves as the high-level control interface for the entire experiment, initiating the evolutionary loop, monitoring its progress, and providing tools for the final analysis of the generated archive.

A complete evaluation cycle, which forms the core feedback loop of the system, proceeds as follows:

1. The **Pyribs Engine** (‘notebook.ipynb‘) ‘asks‘ for a new batch of solutions and sends the corresponding genotypes via a POST request to the API.
2. The **Express API** (‘mapElitesAPI.js‘) receives the request and forwards it to the **Track Generator** module.
3. The generator creates the track phenotype and triggers the evaluation, which starts a fresh, containerized **TORCS** simulation.
4. The custom **Telemetry Module** within TORCS captures gameplay data, which is

processed and saved as structured logs.

5. Finally, the telemetry data, containing the fitness score and behavioral characteristics, is returned to the Pyribs engine, which ‘tells‘ the information to the archive, thus completing the loop.

This integrated pipeline provides a robust and scalable framework for our research, enabling the systematic, automated exploration of the racing track design space.

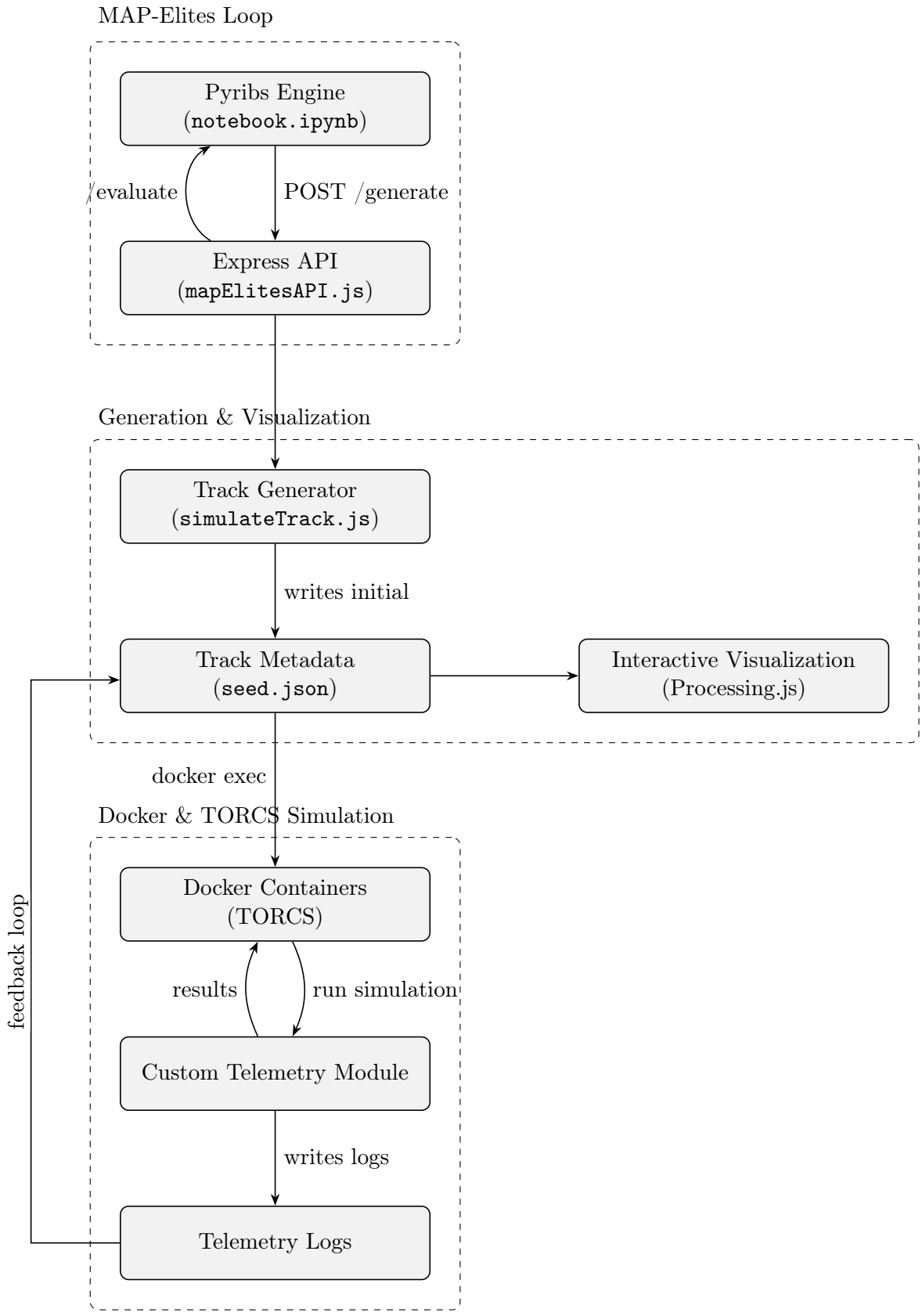

Figure 3.9: The pipeline flows from the Pyribs-driven evolutionary engine to track generation and simulation, with feedback loops completing the evolutionary cycle.

4 | Results and Analysis

This chapter details the experimental work conducted to evaluate the proposed track generation methods. The research was structured in two main phases. The first is a preliminary analysis designed to systematically identify and select robust, meaningful features for guiding the evolutionary search. This foundational work ensures the reliability and interpretability of the final experiments. The second phase involves running the main MAP-Elites experiments using the features selected and validated in the first phase.

4.1. Preliminary Analysis

The objectives of this phase were twofold: firstly, to systematically quantify the stability and reliability (noisiness) of emergent features extracted from TORCS simulation outputs, and secondly, to evaluate how the track generation process and its geometric representations impact feature stability and consistency, which is crucial for meaningful evolutionary comparisons and selections.

These analyses were foundational in ensuring the effectiveness of the descriptors subsequently employed to guide the evolutionary illumination process. Reliable, stable metrics are critical for effectively guiding the evolutionary exploration and accurately characterizing track behavior.

Results were systematically evaluated for both track representations.

4.1.1. Emergent Features Noisiness Analysis

Emergent features extracted from TORCS simulations—such as lap times, overtaking occurrences, and driving patterns—exhibit inherent variability even under conditions of minimal or visually negligible changes in track layouts. While the underlying TORCS physics engine operates deterministically, the complex, adaptive behaviors of AI agents introduce practical non-determinism into the emergent gameplay metrics. Topological track features, derived directly from the track's geometry, remain constant for a given map. However, emergent features, which result from the interaction of agents with the track, can vary across different simulation runs.

This variability is a significant challenge for search-based algorithms like MAP-Elites. If a feature fluctuates wildly on the same track, it cannot be a reliable indicator of quality. The algorithm would optimize random noise rather than meaningful design, undermining the search process. Therefore, quantifying feature reliability is essential.

To quantify the inherent variability (or *noisiness*) of emergent features, we performed controlled experiments involving multiple repeated simulations for each track representation. We designed a systematic approach to assess the stability of emergent features across multiple simulation runs.

Our methodology began by generating a set of 100 representative track seeds for each encoding method (Voronoi and Convex Hull), providing a comprehensive basis for comparative analysis between the two track representations.

Initially, we attempted to modify the starting positions of vehicles on otherwise identical tracks. However, this strategy proved problematic, as even small changes in initial positioning drastically altered race dynamics—affecting collision probabilities and overtaking opportunities—resulting in excessively high variability that obscured meaningful measurements.

Instead, we adopted a more controlled approach by leveraging the mutation operator described earlier to create subtly different track variants. These minimally-mutated tracks were visually indistinguishable to human observers yet contained enough variation to mimic realistic evolutionary perturbations. For each of the 100 original tracks, we generated mutated variants and conducted simulations to quantify the stability of emergent features under these controlled conditions.

This approach was based on the principle that slightly different tracks should yield similar emergent features, thus confirming the robustness of the descriptors against minor geometric perturbations.

The analysis focused on several key emergent features, including lap times, overtaking counts, speed and acceleration entropies, and positional discrepancies (*deltaX*, *deltaY*, *deltaAngleDegrees*). These features were selected for their relevance to both track geometry and gameplay dynamics.

We employed statistical measures such as Coefficient of Variation (CV) and Signal-to-Noise Ratio (SNR) to assess feature reliability. For a given feature with a mean μ and standard deviation σ across repeated simulations, these metrics are defined as:

$$\text{CV} = \frac{\sigma}{\mu} \tag{4.1}$$

$$\text{SNR} = \frac{\mu}{\sigma} \tag{4.2}$$

A low CV and a high SNR indicate a more stable and reliable feature, making it a suitable descriptor for our evolutionary algorithm. This rigorous process confirmed the importance of ensuring that our chosen descriptors represent stable track characteristics rather than simulation artifacts or noise.

The noisiness analysis revealed several critical insights into the stability of emergent features across different track representations.

A particularly significant finding involved the *total_overtakes* metric. While this metric has intuitive appeal as a measure of competitive dynamics, our analysis revealed it was

highly sensitive to minor positional discrepancies at track start and end points (*deltaX*, *deltaY*). These positioning errors could artificially inflate overtaking counts when vehicles stalled or slowed significantly at race onset due to imperfect track closure.

To address this limitation, we developed a normalization strategy that divides *total_overtakes* by the positional mismatch (delta error). This normalized metric (*total_overtakes / delta*) proved more reliable for capturing genuine competitive dynamics. It effectively emphasized overtakes occurring under stable racing conditions while penalizing scenarios where high overtake counts resulted from positioning artifacts rather than track design quality.

To summarize, the noisiness analysis was conducted in two main phases:

Convex Hull Technique. This encoding demonstrated notably high stability for certain geometric metrics, particularly *length* (CV = 0.045, SNR = 22.40), *avg_radius_mean* (CV = 0.081, SNR = 12.38), and *avg_radius_var* (CV = 0.093, SNR = 10.71). Such high signal-to-noise ratios indicate minimal variability, confirming their suitability as stable descriptors. Conversely, metrics such as positional mismatches and traffic-related features displayed substantial instability. The most problematic metrics were those related to track closure error and race gaps, including *deltaY* (CV = 11.73), *deltaAngleDegrees* (CV = 10.06), and *gaps_var* (CV = 9.31). This instability makes them poor candidates for primary behavioral descriptors without normalization. The robustness of these metrics for the Convex Hull technique is visually summarized in Figure 4.1.

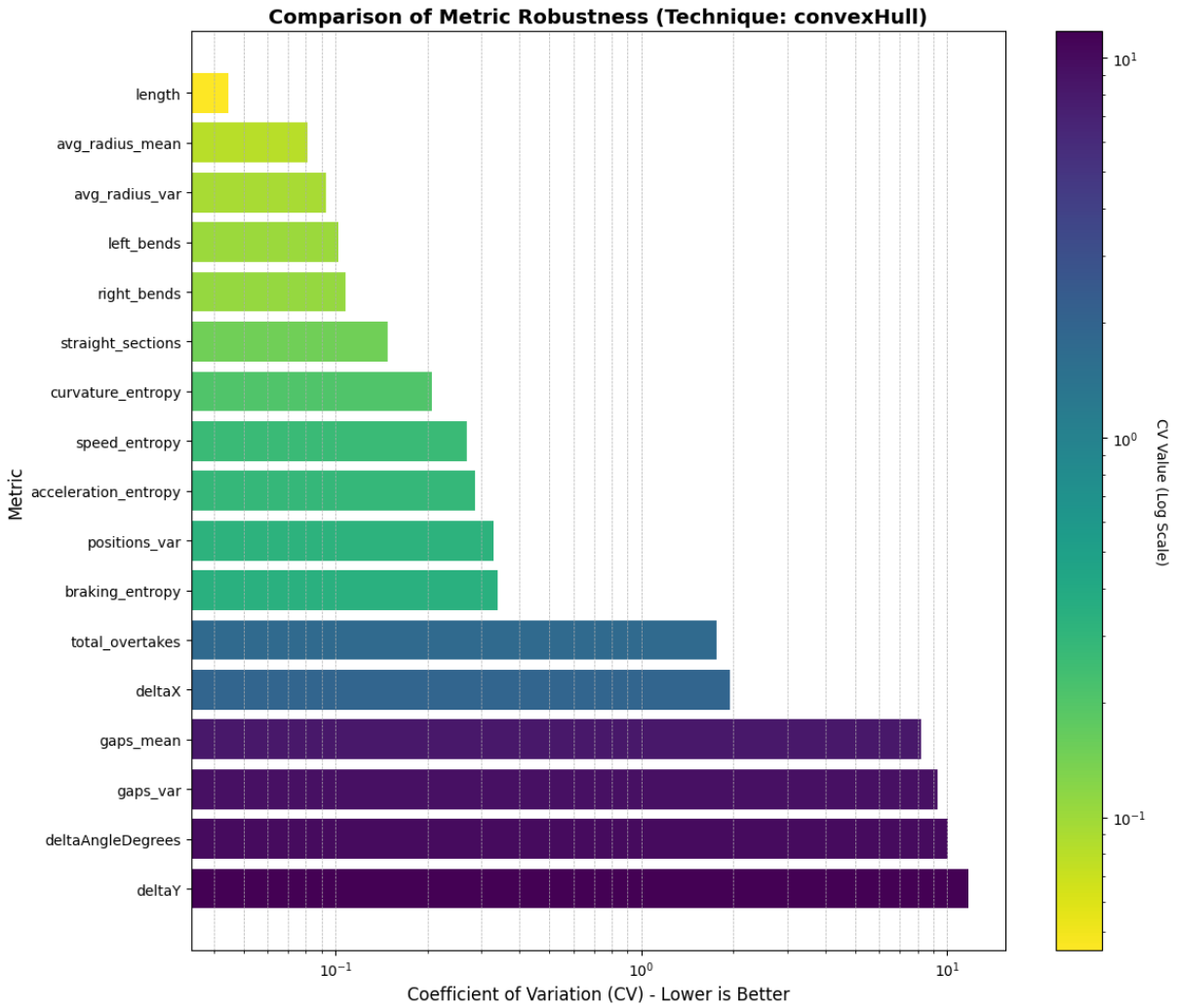

Figure 4.1: Comparison of Metric Robustness for Convex Hull tracks, showing Coefficient of Variation (CV) for various emergent features. Lower CV indicates higher robustness.

Voronoi Technique. Overall, the Voronoi technique produced tracks with higher feature variability compared to the Convex Hull method. However, several key features demonstrated acceptable stability, confirming their viability as behavioral descriptors. The most reliable were *avg_ radius_ mean* (CV = 0.176, SNR = 5.68) and *curvature_ entropy* (CV = 0.198, SNR = 5.05). In contrast, many metrics, particularly those related to positional discrepancies, were highly unstable. These included *deltaAngleDegrees* (CV = 26.12) and most notably *deltaY* (CV = 37.28), highlighting their extreme sensitivity to minor geometric perturbations inherent in the Voronoi generation process. The results for the Voronoi technique are presented in Figure 4.2.

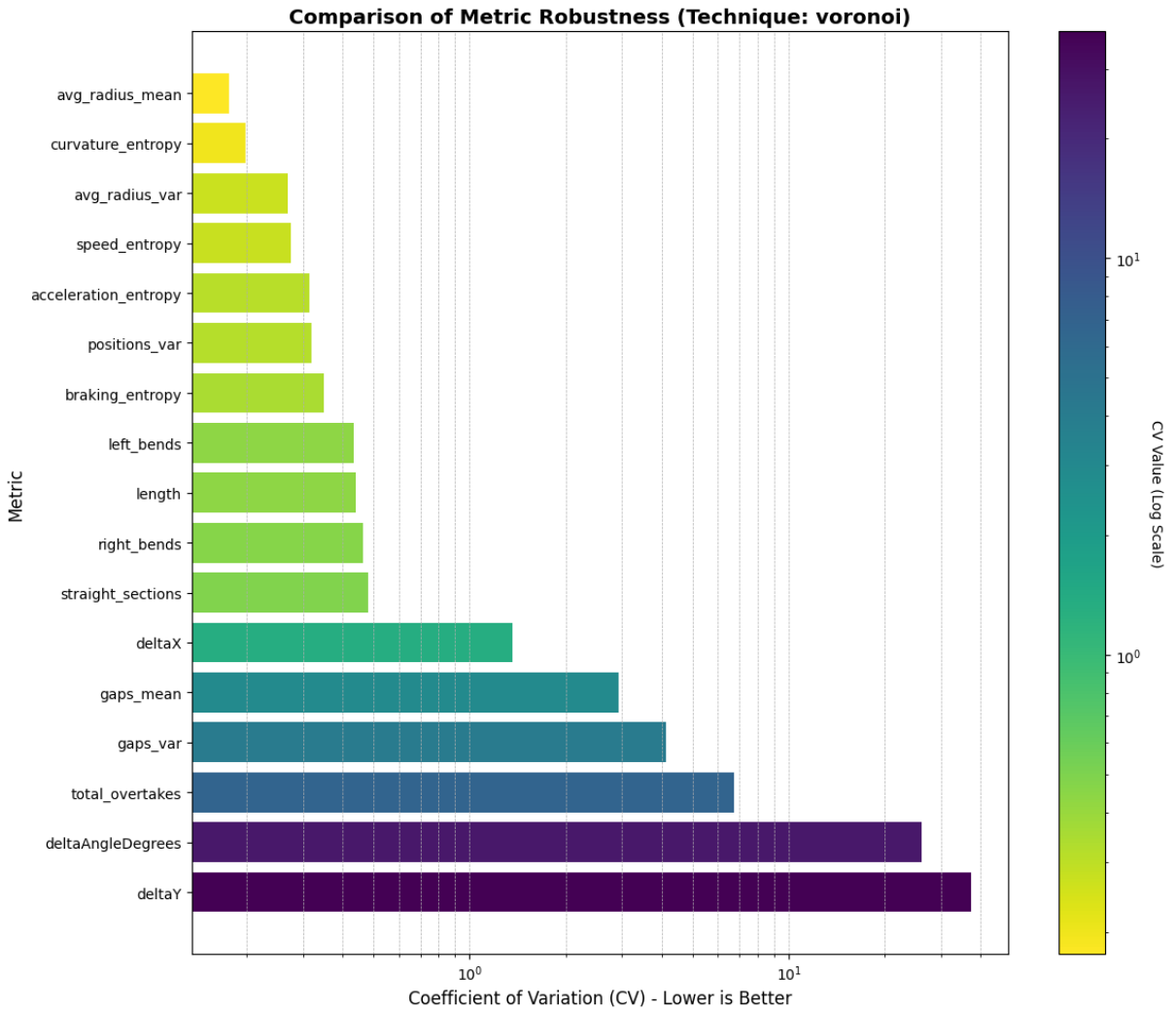

Figure 4.2: Comparison of Metric Robustness for Voronoi tracks, showing Coefficient of Variation (CV) for various emergent features. Lower CV indicates higher robustness.

The outcome of this analysis informed the selection of robust behavioral descriptors and the refinement of the fitness function by identifying and mitigating unreliable metrics, thereby enhancing the stability of the evolutionary search.

These critical insights from the noisiness analysis directly informed two subsequent methodological decisions:

- **Behavioral Descriptor Selection:** Features demonstrating high stability and low redundancy (as confirmed by subsequent correlation analysis) were prioritized as robust behavioral descriptors for the MAP-Elites algorithm. Conversely, highly unstable or redundant metrics were either excluded or refined (as in the case of normalized overtakes) to ensure that the evolutionary search was guided by reliable characteristics.

- **Fitness Function Refinement:** The understanding of feature reliability led to the careful construction of the fitness function. Metrics identified as noisy or susceptible to artifacts were either normalized or weighted appropriately. This ensured that the fitness function accurately reflected genuine track quality, thereby enhancing the stability and effectiveness of the evolutionary search process.

Having established a set of reliable features through the noisiness analysis and developed mitigation strategies for unstable metrics, the next logical step was to investigate the relationships among these validated features. While a feature may be reliable, it might also be highly correlated with another, making it redundant. The subsequent correlation analysis aims to prune this set of reliable features further by identifying and removing such redundancies. This ensures the final set of descriptors for MAP-Elites is not only stable but also efficient.

4.1.2. Correlation and Clustering Analysis of Metrics

Informed by the noisiness analysis, the second phase of the preliminary study involved in-depth correlation and hierarchical clustering analyses of the averaged metrics across all repeated simulations. Pearson correlation coefficients were calculated for all pairs of metrics for both the Convex Hull and Voronoi generation techniques, as visualized in Figures 4.3 and 4.4.

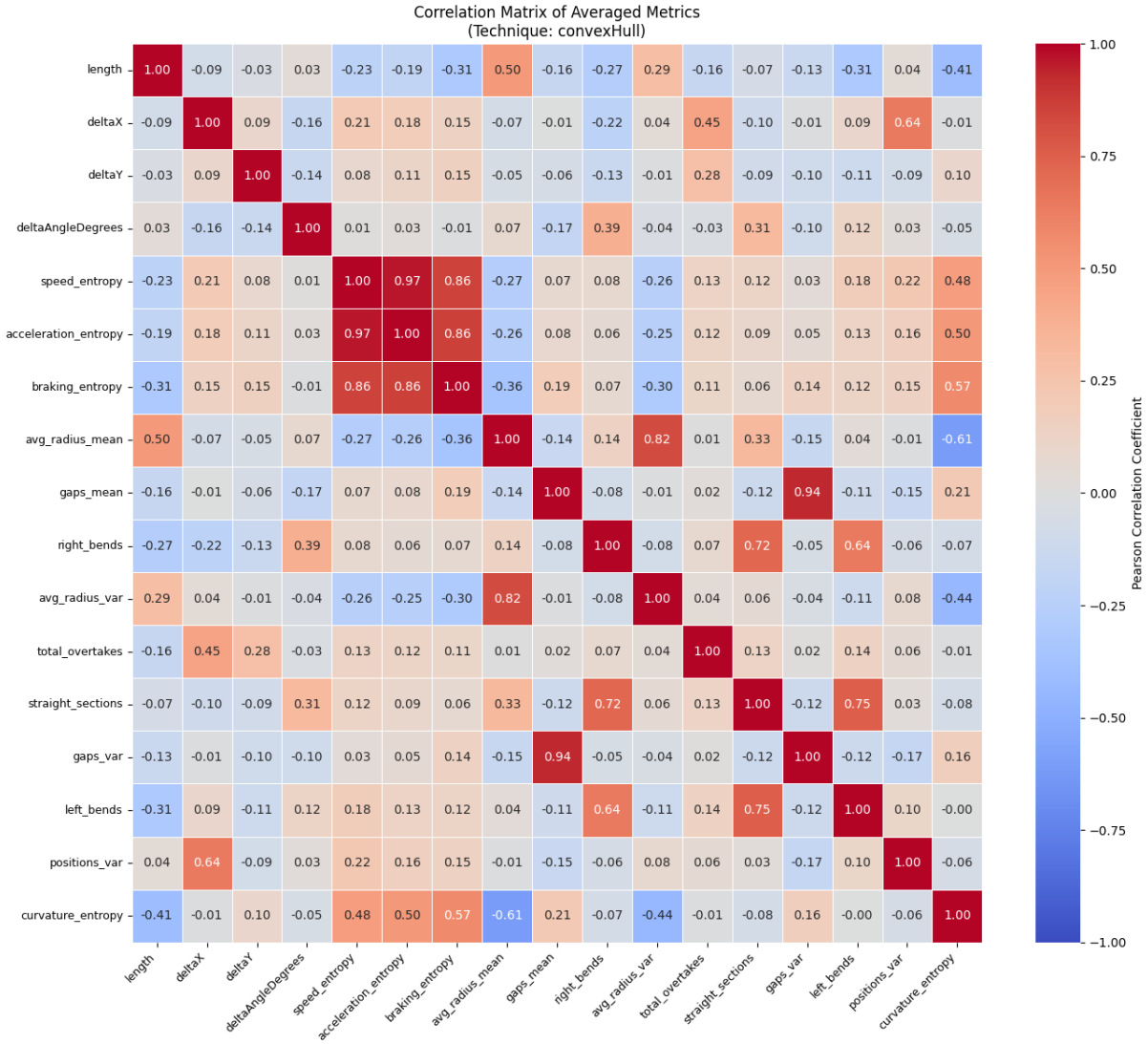

Figure 4.3: Correlation matrix for Convex Hull features.

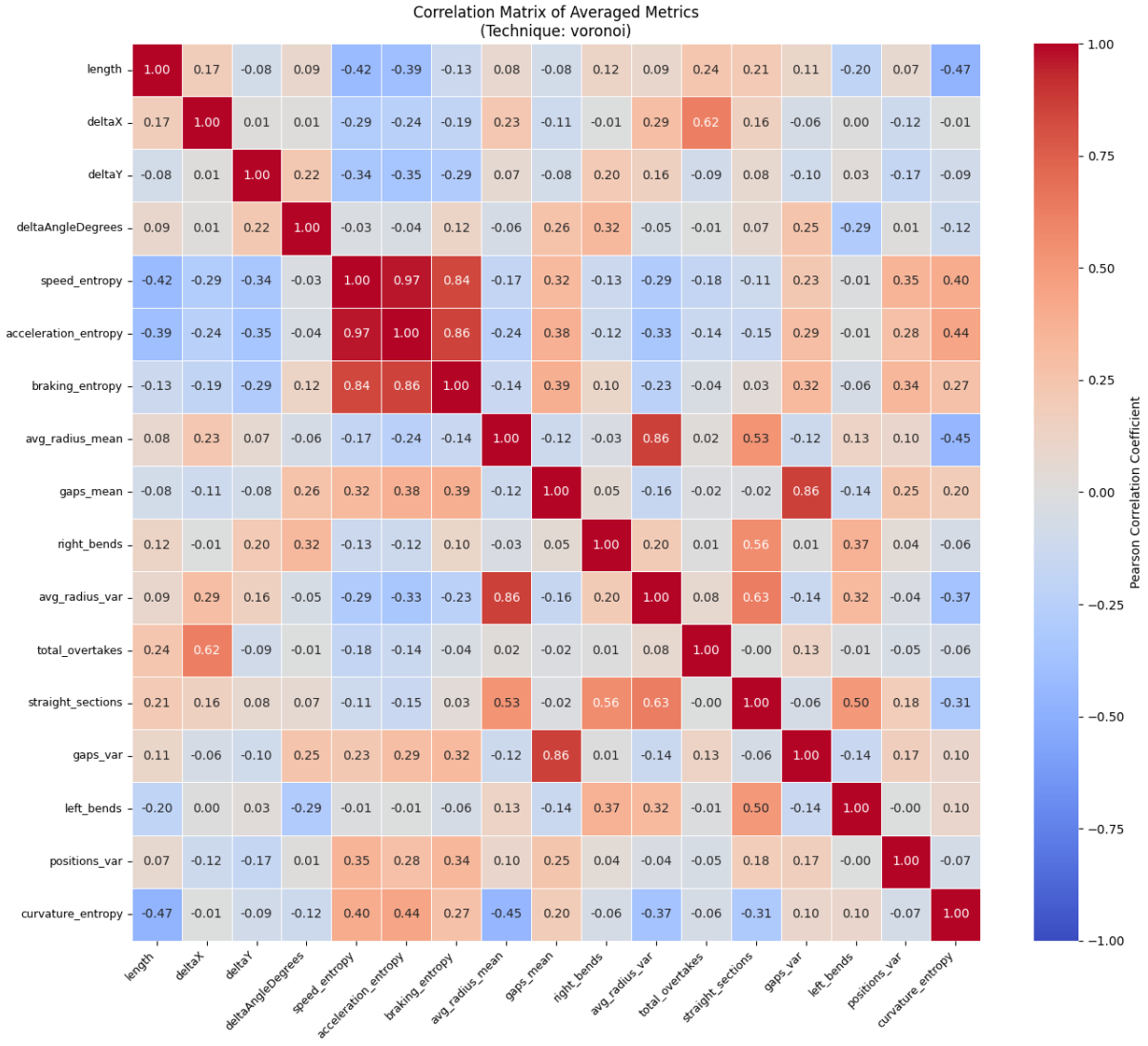

Figure 4.4: Correlation matrix for Voronoi features.

A direct comparison of the two correlation matrices reveals several key structural patterns and redundancies in the feature space. For both generation techniques, a tight **"driving dynamics" cluster** is immediately apparent, with `speed_entropy`, `acceleration_entropy`, and `braking_entropy` exhibiting very high positive correlations (consistently $r \geq 0.84$). This indicates that these three metrics largely capture the same underlying concept of driving complexity. Similarly, `gaps_mean` and `gaps_var` are strongly correlated ($r \approx 0.94$ for Convex Hull, $r \approx 0.86$ for Voronoi), suggesting they are functionally redundant.

Critically, the matrices provide quantitative evidence for the issues identified in the noisiness analysis. For Voronoi tracks, there is a notable correlation between track closure error (`deltaX`) and `total_overtakes` ($r = 0.62$). This statistically supports the hypothesis that poorly closing tracks can artificially inflate overtaking counts due to simulation artifacts. For Convex Hull tracks, a similar, though weaker, relationship exists ($r = 0.45$). These findings underscore the necessity of using a normalized overtaking score to ensure

a reliable fitness metric.

Further analysis using hierarchical dendrograms (4.5, 4.6) confirms these groupings. Based on this analysis, several conclusions were drawn to guide the selection of behavioral descriptors:

1. **Driving Dynamics Cluster.** The strong correlations between the three entropy-based driving metrics ($r \geq 0.84$) confirm their redundancy. Selecting a single representative, such as *speed_ entropy*, is sufficient to capture the complexity of vehicle dynamics without cluttering the behavioral space.

2. **Redundancy in Gameplay Metrics.** The extremely high correlation between `gaps_mean` and `gaps_var` ($r > 0.85$) makes them interchangeable. More importantly, the observed link between `deltaX` and `total_overtakes` confirms that a naive overtaking metric is confounded by simulation errors. This reinforces the decision to engineer a normalized metric ($overtakes/\delta$) to disentangle genuine competitive dynamics from artifacts of poor track closure.

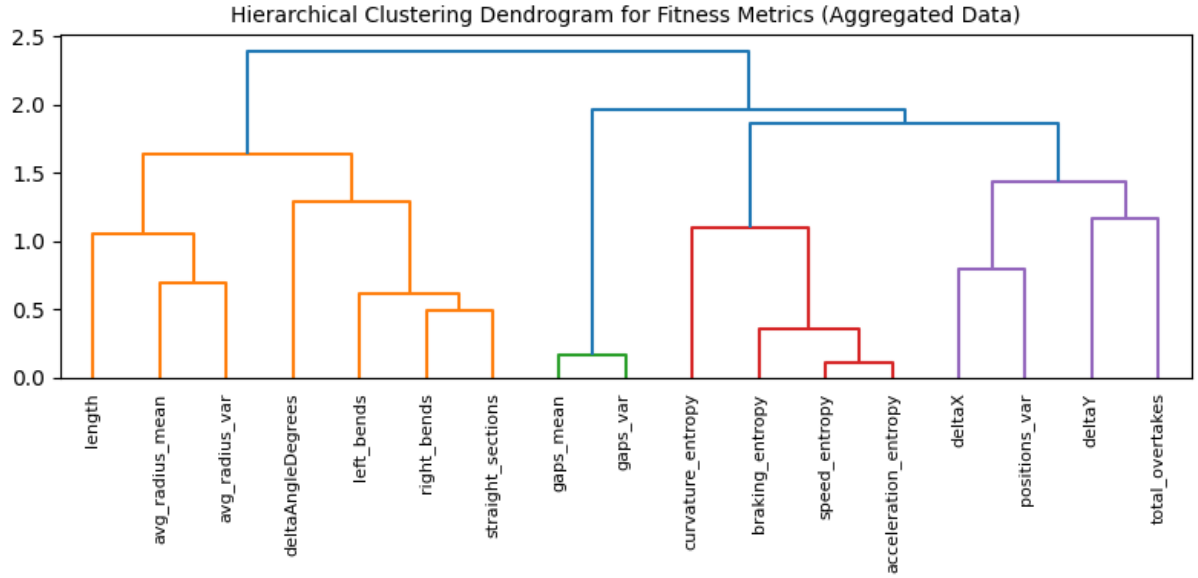

Figure 4.5: Hierarchical clustering dendrogram for Convex Hull track features, illustrating relationships and redundancies among metrics.

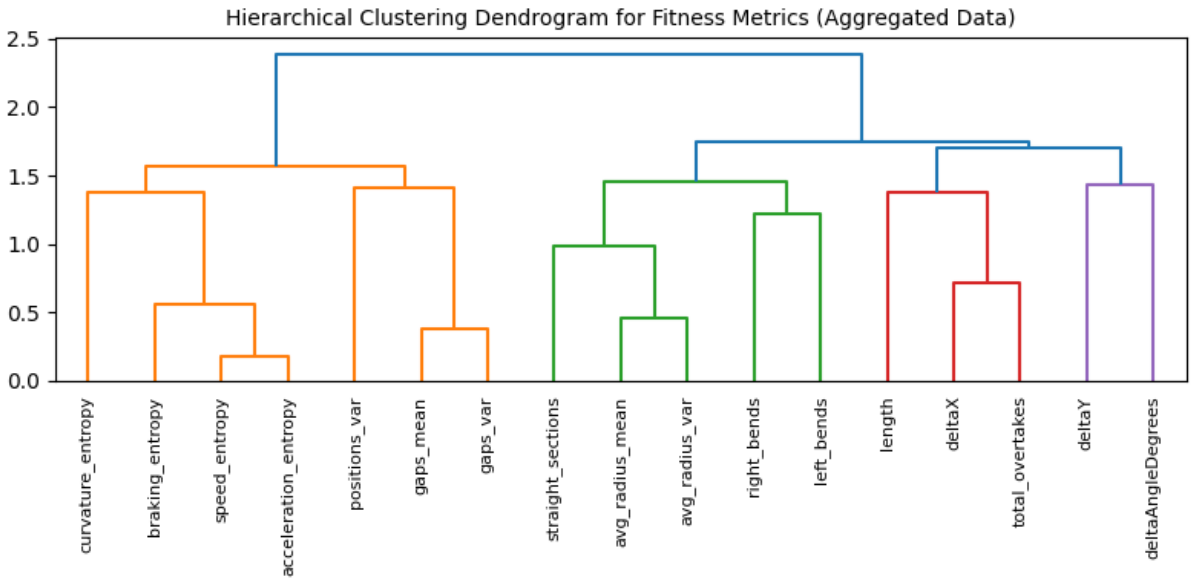

Figure 4.6: Hierarchical clustering dendrogram for Voronoi track features, showing distinct and pronounced clusters.

To conclude these preliminary investigations, the correlation and clustering analyses confirmed the stability of key emergent features while also revealing significant redundancies and confounding factors. This understanding guided the selection of a minimal yet effective set of descriptors for subsequent MAP-Elites experiments.

4.1.3. Dimensionality reduction experiments

After assessing feature stability, this study extracts higher-level behavioral descriptors using dimensionality reduction. This approach utilizes raw track spline data as input for embedding and diverges from methods that rely on track images. Each generated track undergoes sampling to obtain a fixed number of (x,y) coordinate points along its spline. This process forms a high-dimensional vector, which serves as the input for dimensionality reduction algorithms. Specifically, the Uniform Manifold Approximation and Projection (UMAP) and t-distributed Stochastic Neighbor Embedding (t-SNE) algorithms process these high-dimensional spline vectors. These techniques project complex, high-dimensional data into a lower-dimensional space, typically two dimensions, while preserving the underlying structure and local neighborhood relationships of the data points.

Using both algorithms served as a cross-validation method for the resulting low-dimensional space. The algorithms differ in their mathematical objectives and are therefore suited for revealing different aspects of the data's structure. Both algorithms are applied to the same preprocessed spline data. This comparative evaluation allows us to select the most suitable technique for our goal: creating a continuous and interpretable behavioral space for the MAP-Elites algorithm. The ideal algorithm will not only group similar tracks but also organize these groups in a globally meaningful way.

A critical preprocessing step for these spline vectors involved making them invariant to

irrelevant transformations:

- **Centering:** Each track's coordinate set is centered around its mean. This removes translational information, ensuring that two identical tracks positioned differently on the canvas are treated as the same shape.
- **PCA Alignment (Rotation Invariant):** To make the embeddings invariant to rotation, each centered track (represented as a set of points) is aligned using Principal Component Analysis (PCA) via Singular Value Decomposition (SVD). This process finds the principal axis of the track—the direction of maximum variance—and rotates the track so that this axis aligns with a fixed reference direction (e.g., the positive x-axis). To ensure a consistent orientation and avoid 180-degree flips, the sign of the second principal component is checked and, if necessary, the track is reflected. This alignment ensures that tracks with identical shapes but different rotations are mapped to similar points in the embedding space.
- **No Scaling:** Unlike centering and alignment, scaling was intentionally omitted. Scaling would normalize track sizes, potentially discarding meaningful information about a track's intrinsic dimensions, such as its overall length or typical curvature radii, which are crucial for distinguishing different racing experiences.

This preprocessing significantly enhances the stability and interpretability of the embeddings by ensuring that the dimensionality reduction process captures intrinsic geometric characteristics rather than artifacts of positioning or orientation.

The effectiveness of these preprocessing steps is demonstrated through comparative analysis of embeddings generated before and after applying centering and PCA alignment. Figure 4.7 shows the t-SNE embedding of raw, unprocessed spline vectors. In this visualization, tracks that are geometrically identical but differ only in translation or rotation are scattered across distant regions of the embedding space. This separation occurs because the dimensionality reduction algorithm treats positional and orientational differences as meaningful geometric variations, obscuring the true structural similarities between tracks.

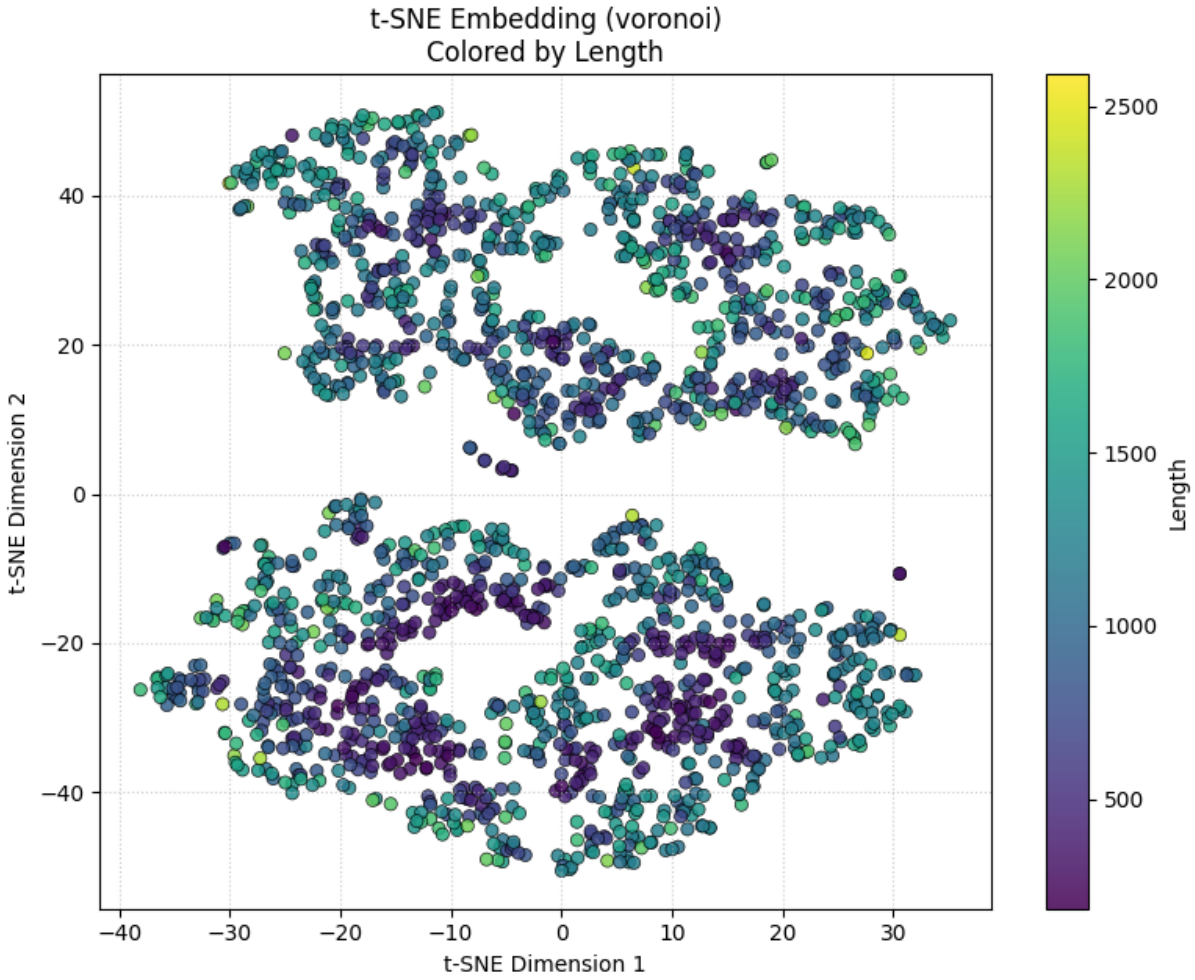

Figure 4.7: t-SNE embedding of raw spline vectors before preprocessing. Geometrically identical tracks with different positions or orientations are erroneously mapped to distant regions, highlighting the critical need for invariant preprocessing.

In contrast, Figure 4.8 presents the t-SNE embedding after applying centering and PCA alignment. The transformation is remarkable; geometrically similar tracks now cluster together regardless of their original position or orientation. The preprocessing successfully eliminates irrelevant transformational variations while preserving meaningful structural relationships. The more symmetric distribution and coherent color gradients in the visualization may indicate that the embedding space now captures intrinsic geometric properties rather than positional artifacts.

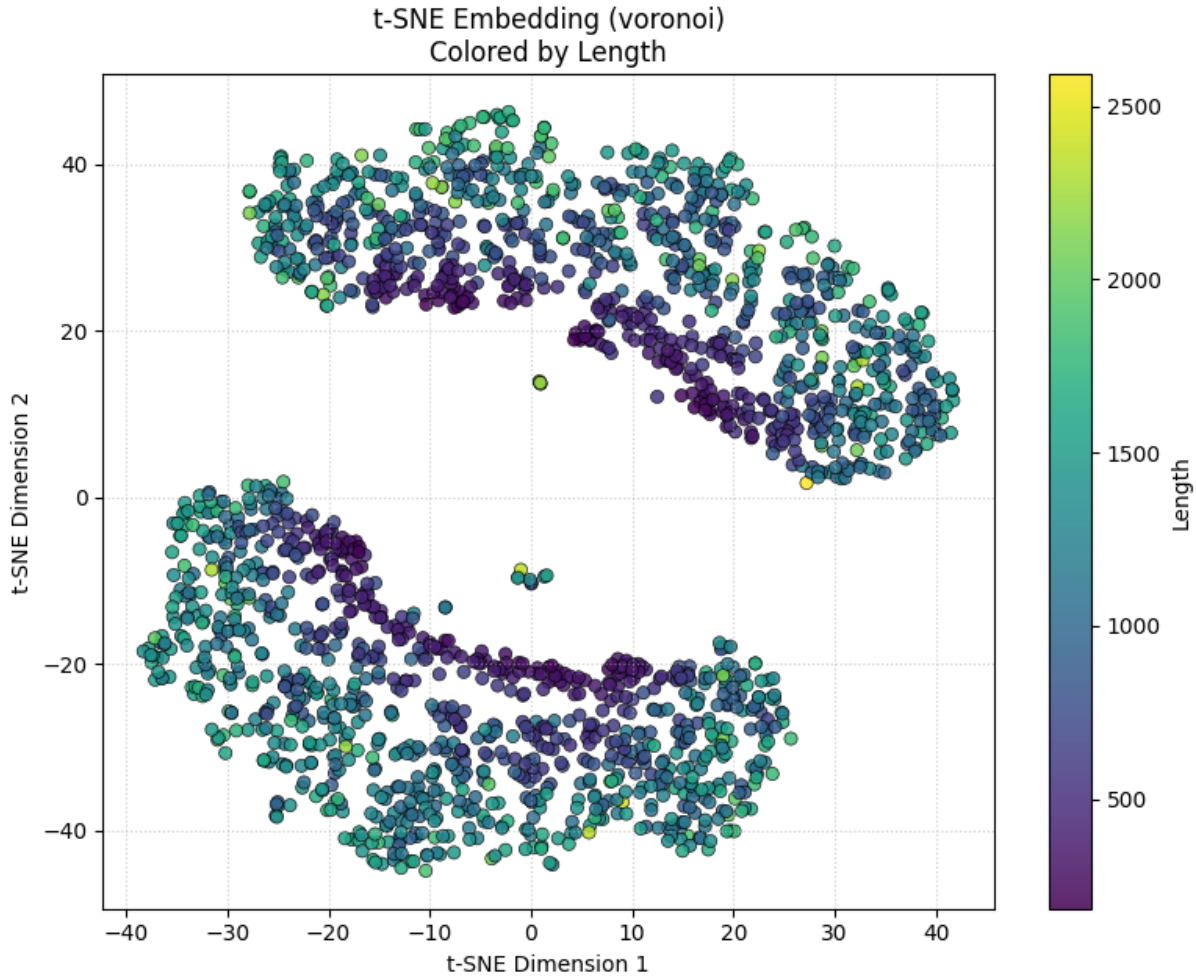

Figure 4.8: t-SNE embedding after centering and PCA alignment. Geometrically similar tracks now cluster together, demonstrating the effectiveness of preprocessing in capturing intrinsic shape characteristics.

The UMAP algorithm exhibits similar improvements through preprocessing, as shown in Figures 4.9 and 4.10.

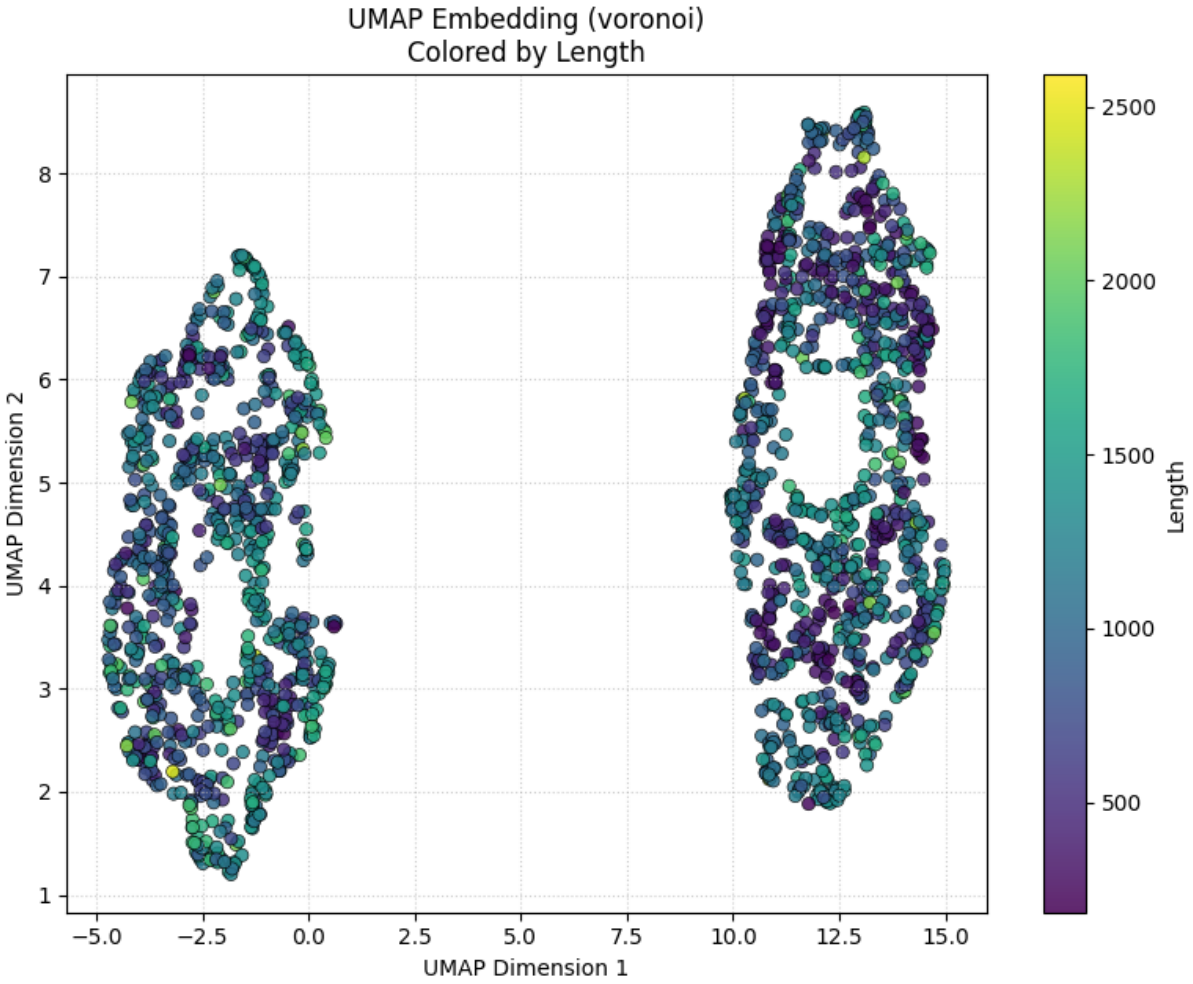

Figure 4.9: UMAP embedding before preprocessing, showing dispersed clustering of geometrically similar tracks.

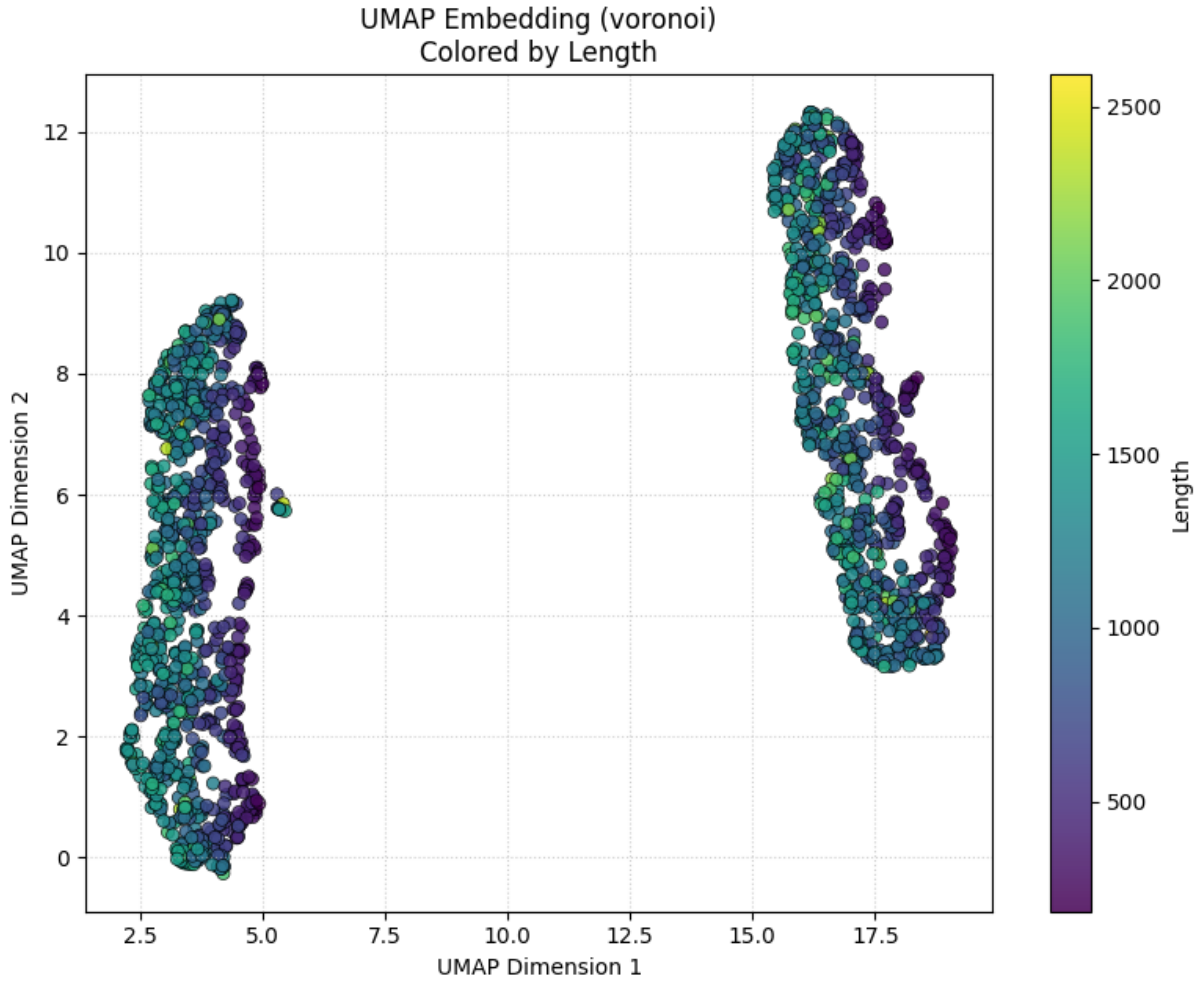

Figure 4.10: UMAP embedding after preprocessing, revealing coherent geometric neighborhoods and improved structural organization.

To conduct a critical methodological validation and obtain diagnostic insights, we analyzed both techniques systematically.

First, the confirmation that track splines can be organized into distinct visual clusters using both algorithms (as shown in Figures 4.8 and 4.10) provides strong evidence that our dataset contains inherent, separable geometric structures. When both t-SNE and UMAP produce similar visual groupings, our confidence in the existence of these groupings as genuine features of the data increases significantly.

Second, the complementary strengths and limitations of t-SNE and UMAP establish a crucial comparative framework. While t-SNE excels at preserving local neighborhoods, it often distorts the global structure of the data. In contrast, UMAP aims to preserve both local and global relationships effectively.

This comparative analysis serves as a validation process that ensures the robustness and validity of our behavioral descriptor generation methodology.

The following figures show the UMAP embedding space colored by different track metrics:

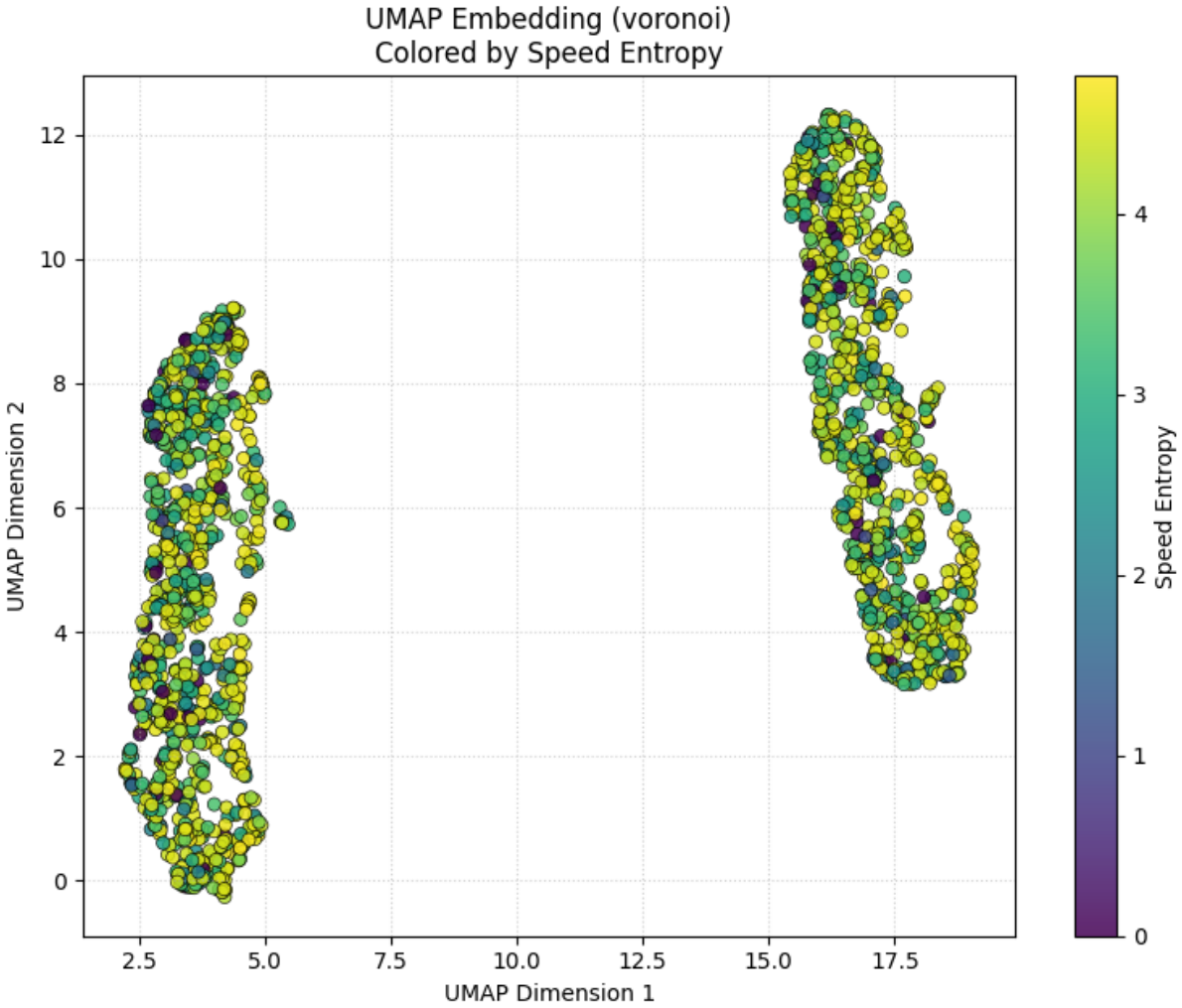

Figure 4.11: UMAP embedding colored by speed entropy.

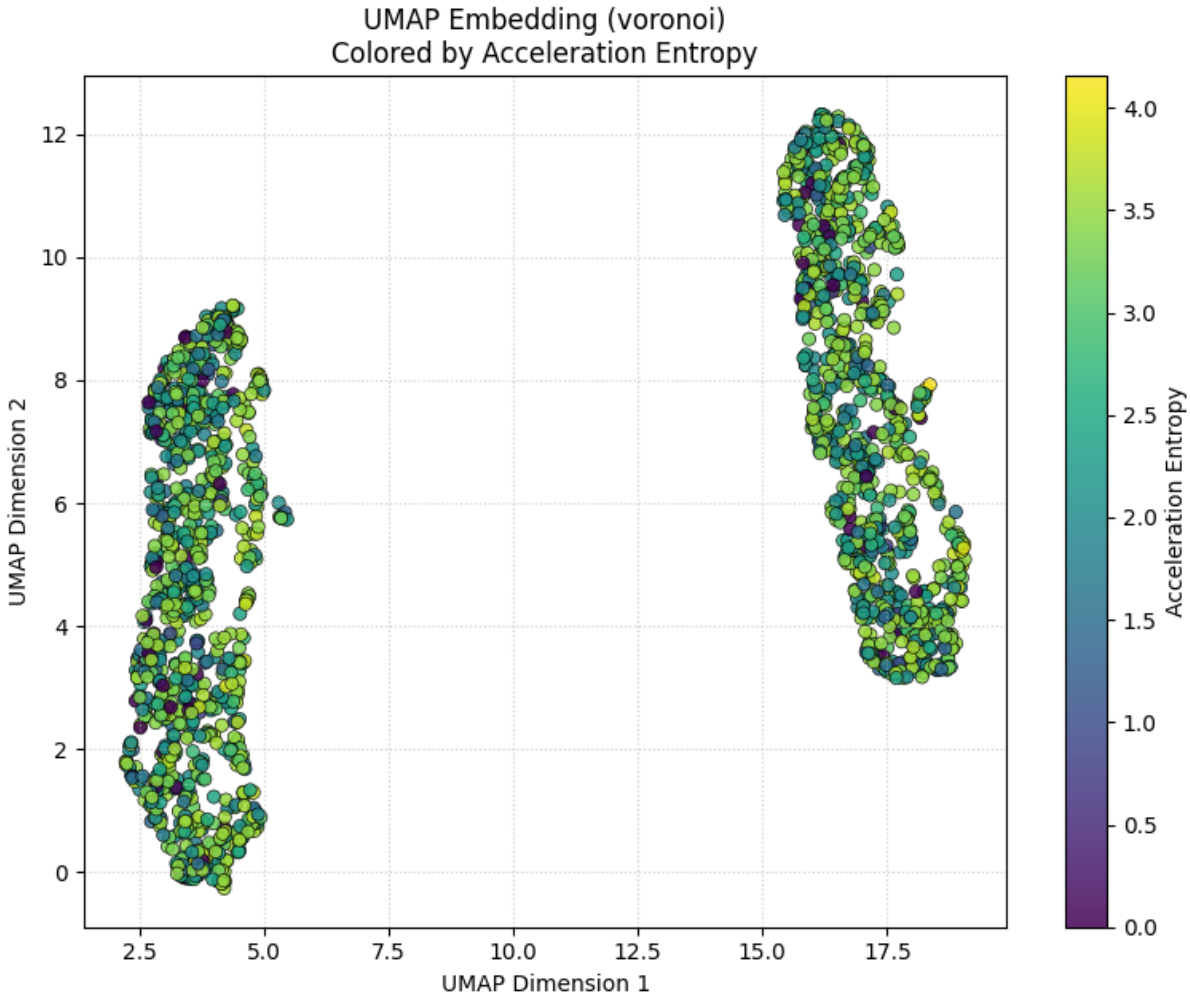

Figure 4.12: UMAP embedding colored by acceleration entropy.

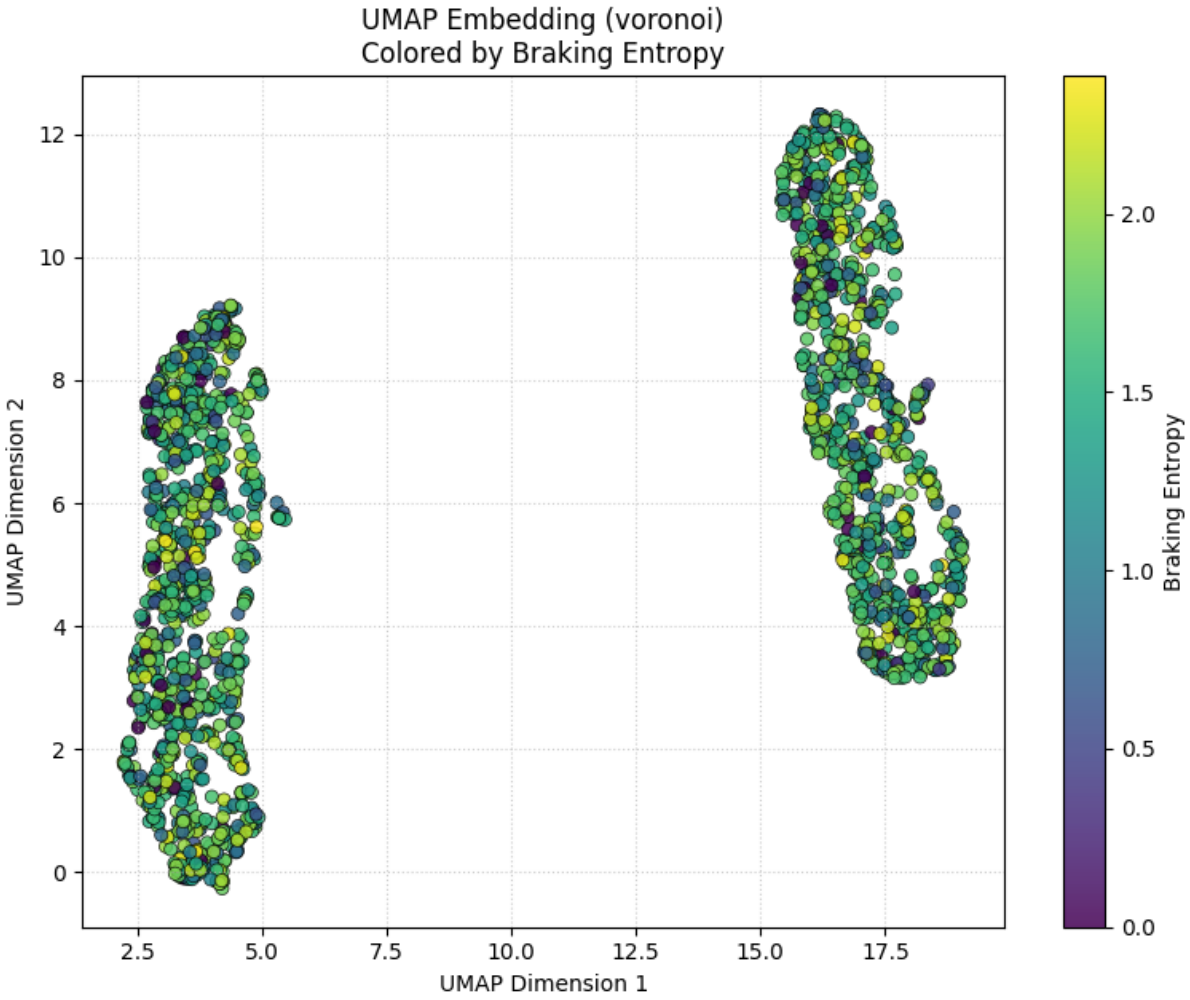

Figure 4.13: UMAP embedding colored by braking entropy.

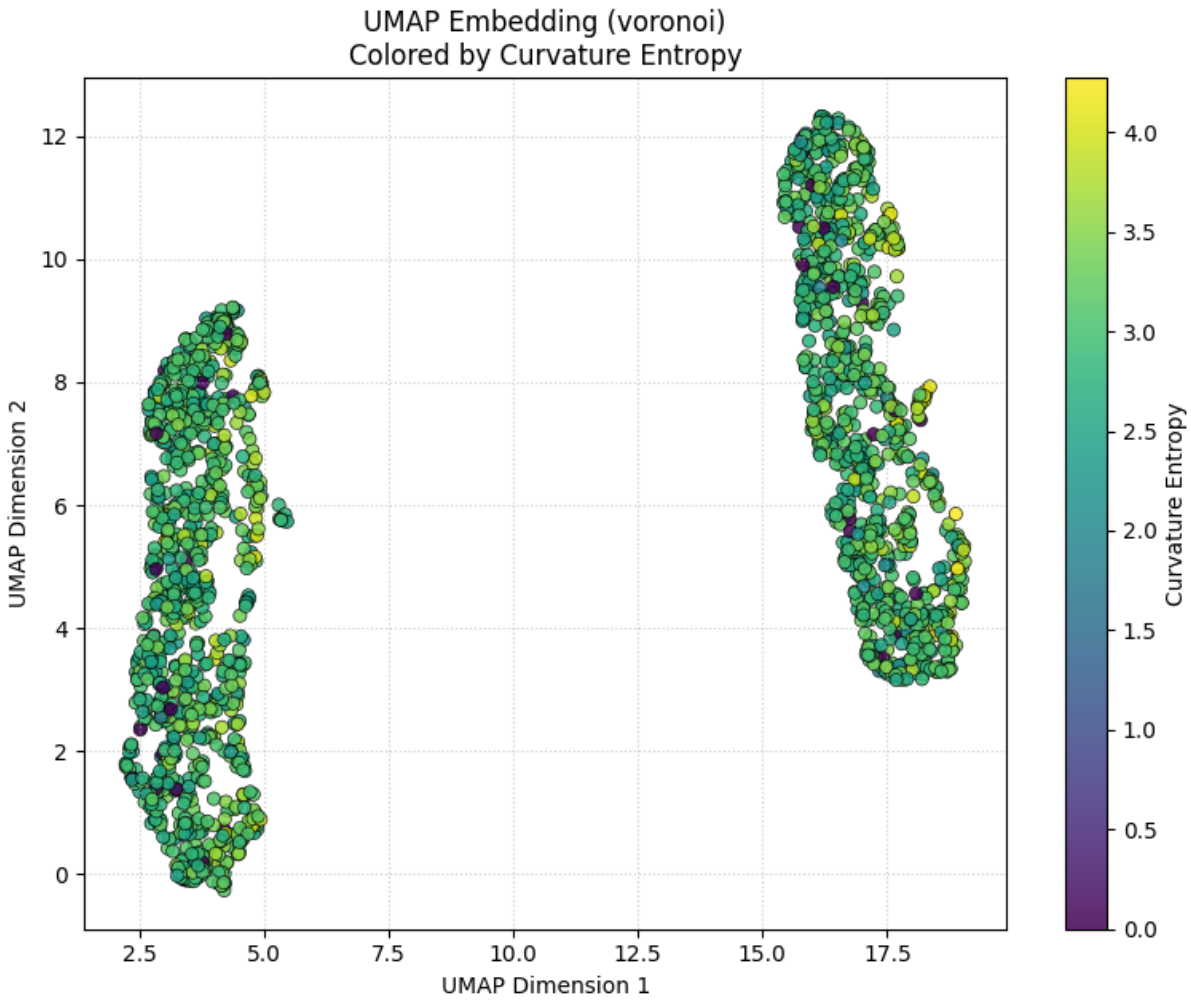

Figure 4.14: UMAP embedding colored by *total_ overtakes*.

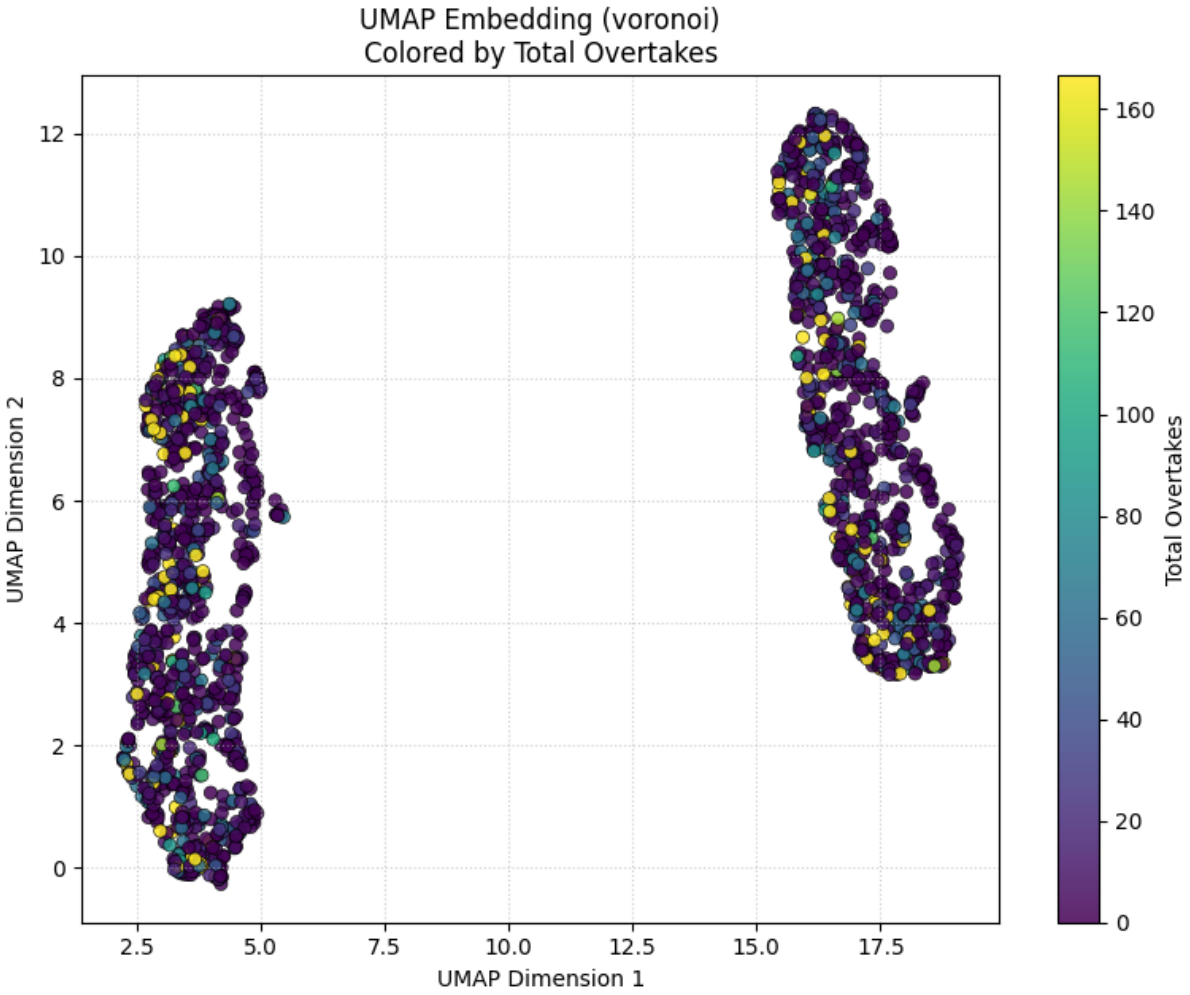

Figure 4.15: UMAP embedding colored by *total_ overtakes*.

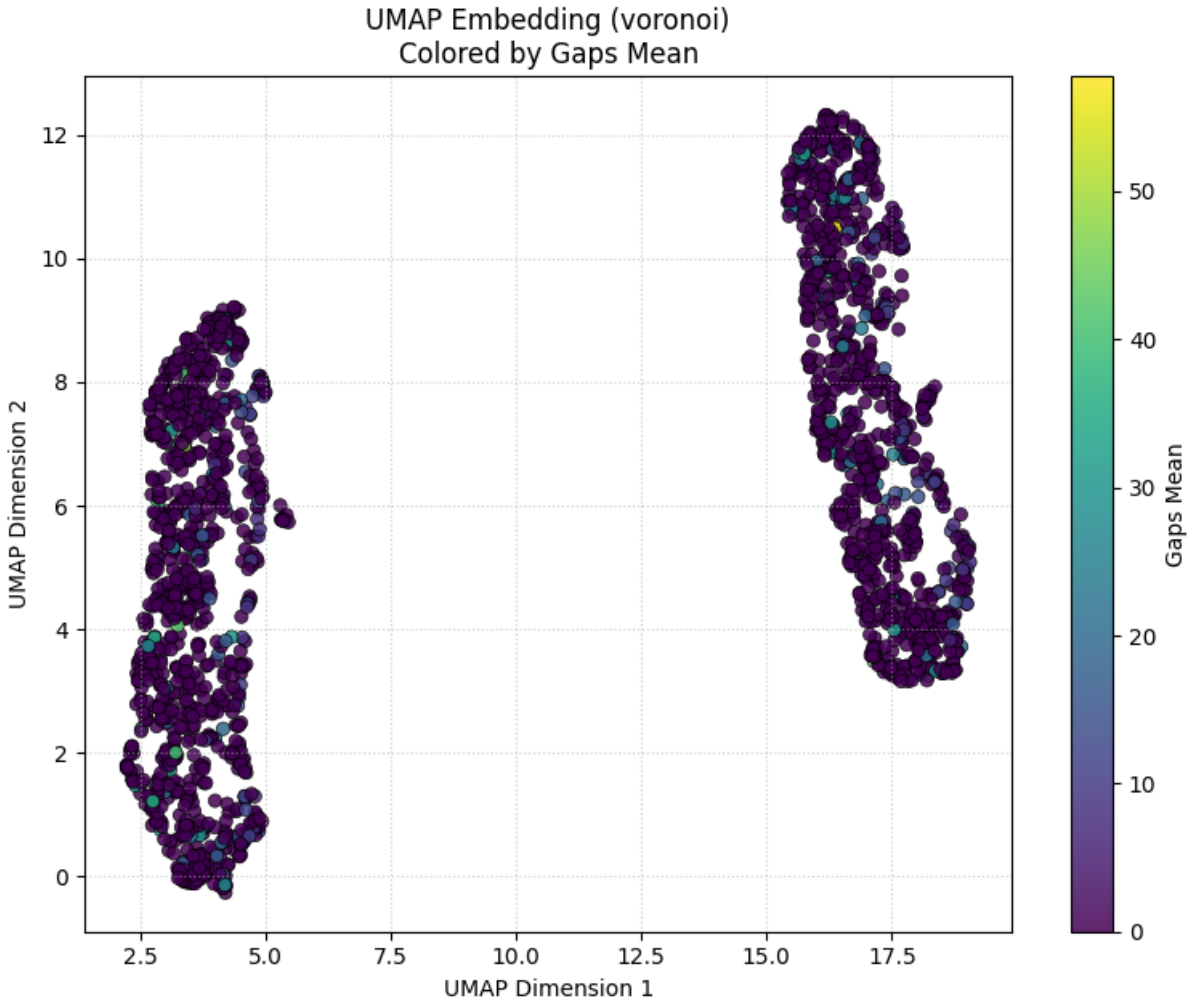

Figure 4.16: UMAP embedding colored by average gap between cars at race end.

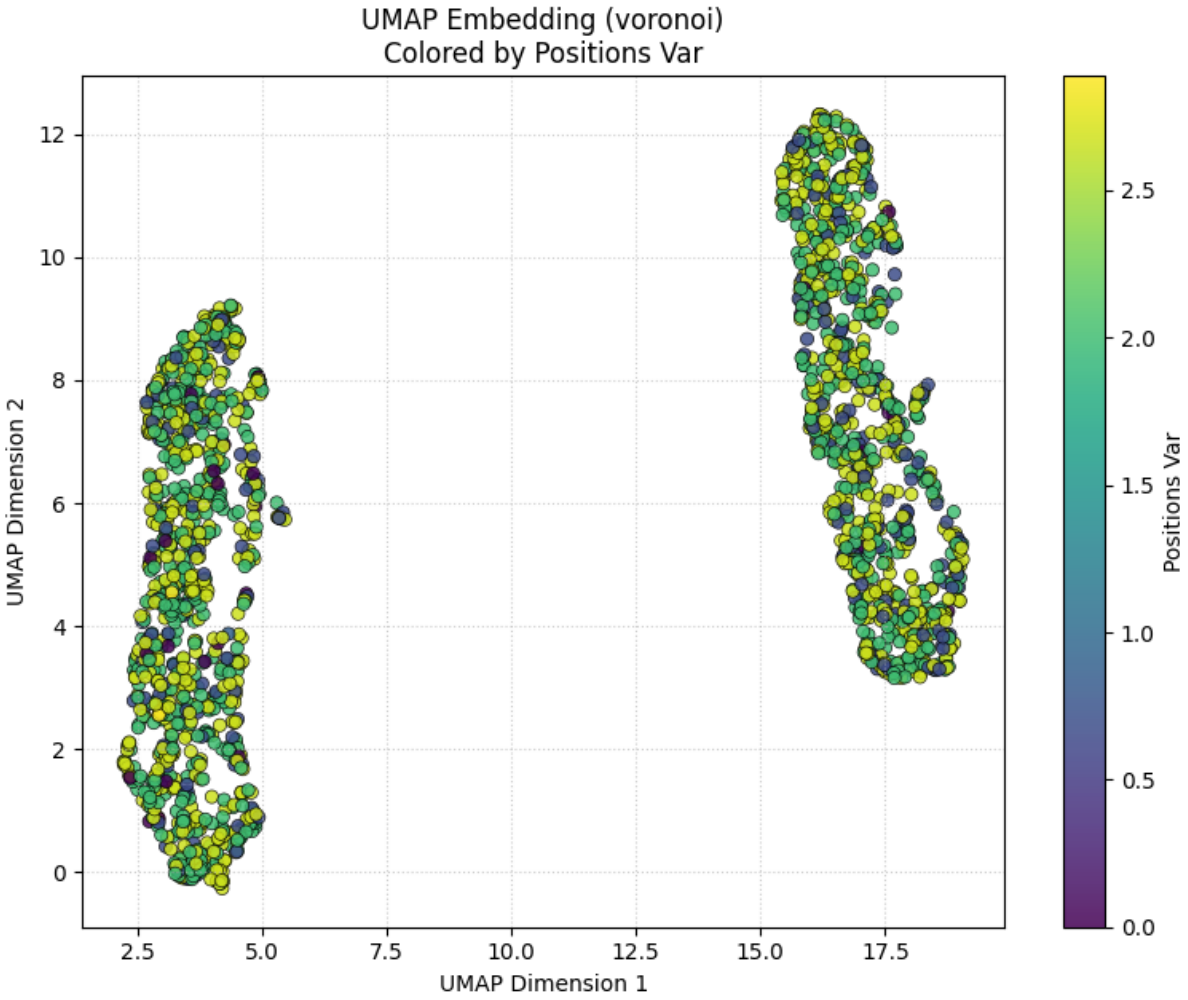

Figure 4.17: UMAP embedding colored by average change in driver positions.

To further illustrate the quality of the resulting embeddings, Figures 4.18 and 4.19 present samples of track neighborhoods extracted from the UMAP embedding space.

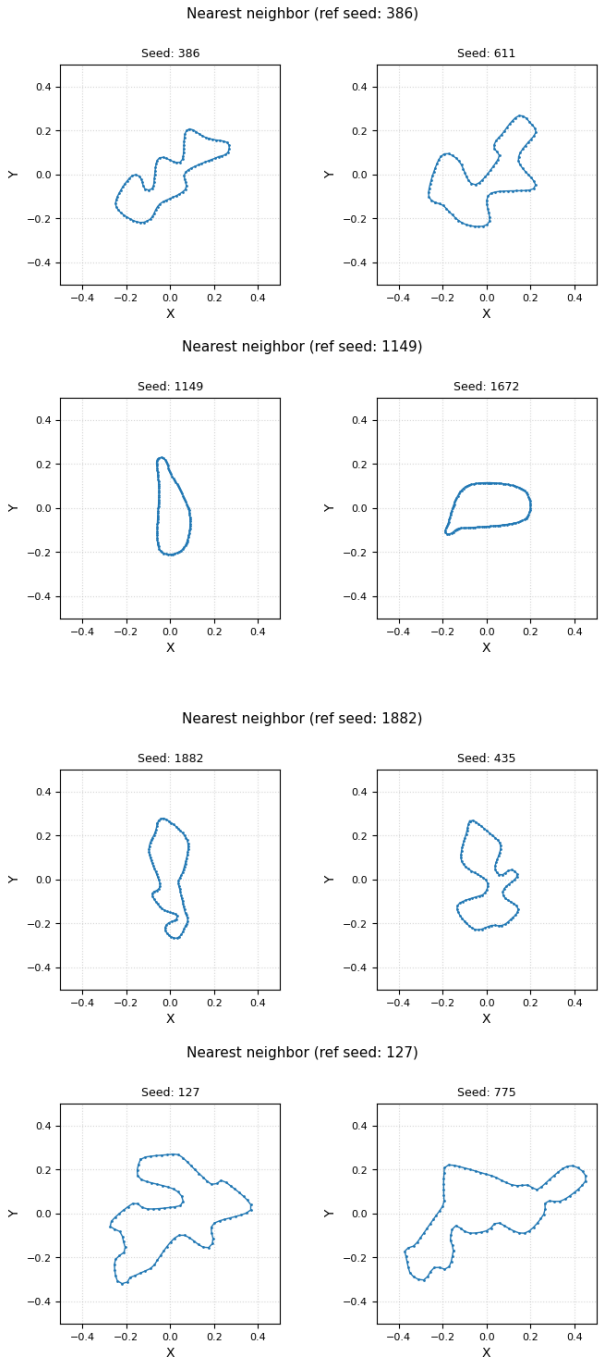

Figure 4.18: Representative track neighborhoods from UMAP embedding space. Tracks within each neighborhood exhibit similar geometric characteristics.

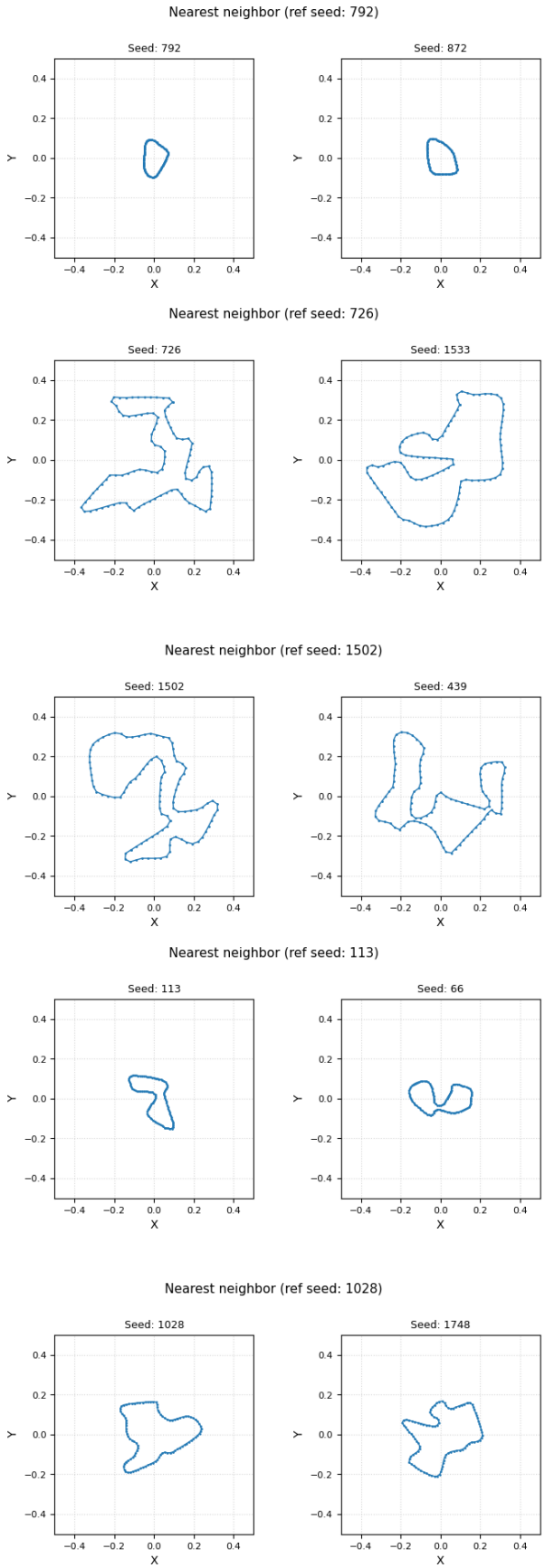

Figure 4.19: Additional track neighborhoods from UMAP embedding space.

These visualizations demonstrate how tracks within the same neighborhood share common geometric characteristics—such as similar curvature patterns, overall shape complexity, or structural motifs—while maintaining distinct individual features. The coherent grouping validates the effectiveness of the preprocessing pipeline and confirms that the embedding space successfully captures meaningful geometric relationships between track designs.

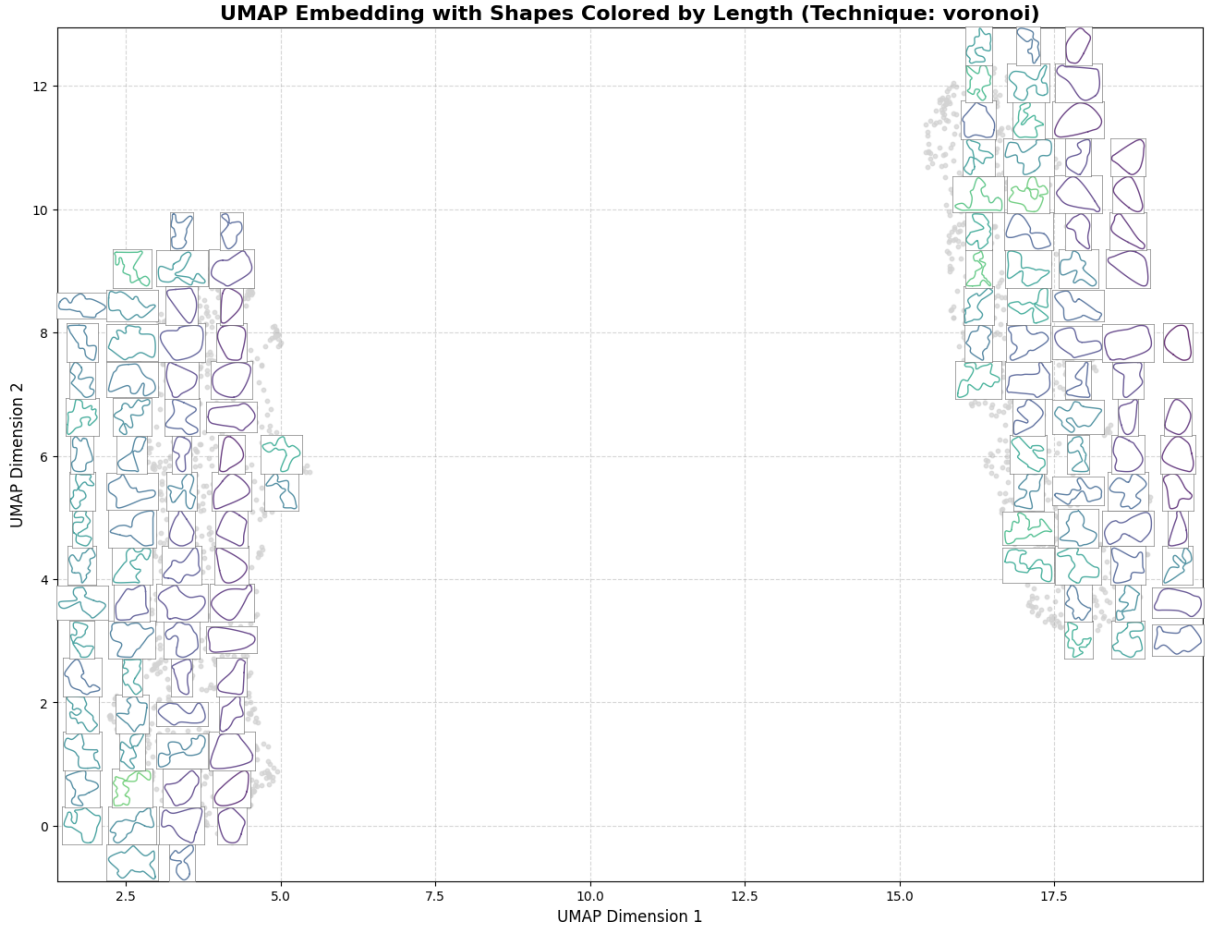

Figure 4.20: UMAP embedding with sample tracks produced using the Voronoi technique from different regions of the embedding space.

A comparison between the two generation methods reveals the superior expressive power of the Voronoi representation. As Figure 4.20 illustrates, the Voronoi technique populates a broad and geometrically diverse region of the embedding space. By contrast, the Convex Hull technique produced a visibly smaller and less varied archive, resulting in a more constrained embedding, as shown in Figure 4.21. This visually confirms that the Voronoi method is more effective at exploring a rich design space.

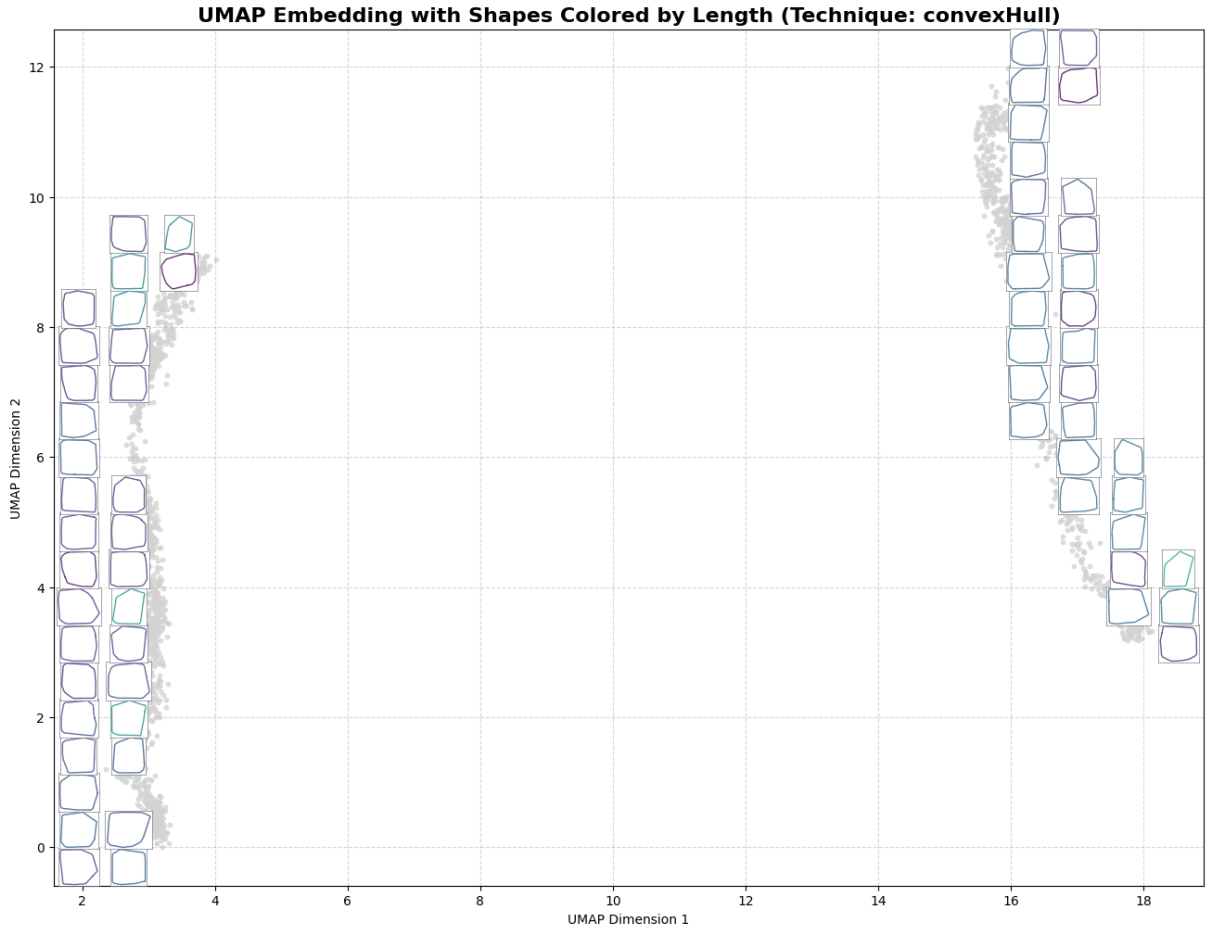

Figure 4.21: UMAP embedding with sample tracks produced using the Convex Hull technique, showing a more constrained design space compared to the Voronoi method.

Given the practical advantages for transforming unseen data, UMAP was selected as the exclusive method for generating behavioral descriptors later in the experiments. The t-SNE analysis served its purpose as a crucial cross-validation step, confirming the existence of local geometric structures and ultimately highlighting UMAP's strengths. This process ensures the robustness and validity of our methodological choice. with UMAP is designed not only to fit a manifold but also to transformational new, unseen data points into the learned embedding space. t-SNE, by contrast, is primarily a visualization technique and lacks a direct, built-in mechanism for this task; projecting new data requires non-trivial extensions. This capability is highly valuable for our generative system. It means that once the UMAP model is trained, it can be saved and reused to quickly characterize any newly generated track by projecting its spline into the existing behavioral space without needing to retrain the entire model.

But before diving into the experiment chapter, we also tried to analyze the two large clusters and the interpretability of the embeddings generated by both UMAP and t-SNE. The next subsection explores whether the visual clusters observed in the embeddings correspond to meaningful differences in track characteristics, and how these dimensions

relate to explicit track metrics.

Clusters and Interpretability

UMAP embeddings consistently displayed distinct visual groups, as presented in the Figures 4.9 and 4.10. This section investigates whether these apparent visual clusters correspond to meaningful differences in track characteristics.

To quantify the significance of these visual groupings, we applied K-Means clustering with two clusters to both t-SNE and UMAP embeddings. Remarkably, both dimensionality reduction techniques produced identical cluster assignments, suggesting consistent underlying structure in the data.

Statistical analysis revealed that while clusters were visually distinct, they exhibited only modest differences in measurable track properties. For instance, t-SNE clusters showed slight variations in track length (1043.4m vs. 1017.4m), right bends, and left bends. However, these differences, while statistically detectable, represent minor practical distinctions that would be barely perceptible to players.

Correlation Analysis and Interpretability To better understand what the embedding dimensions capture, we examined the Spearman correlations between the embedding coordinates and our explicit track metrics. The analysis showed that UMAP Dimension 1 captures a weak signal related to track scale and complexity, evidenced by a moderate negative correlation with *length* ($r = -0.34$). Weaker positive correlations also appeared with entropy-based metrics like *curvature_ entropy* ($r = 0.14$) and *speed_ entropy* ($r = 0.11$). However, as visualized in Figure 4.23, most other metrics showed negligible correlation, and UMAP Dimension 2 appeared to capture no single, interpretable feature. The t-SNE embeddings (Figure 4.22) showed even weaker correlations across the board, with the strongest being only $r = -0.07$ with *length*, reinforcing the challenge of interpretation.

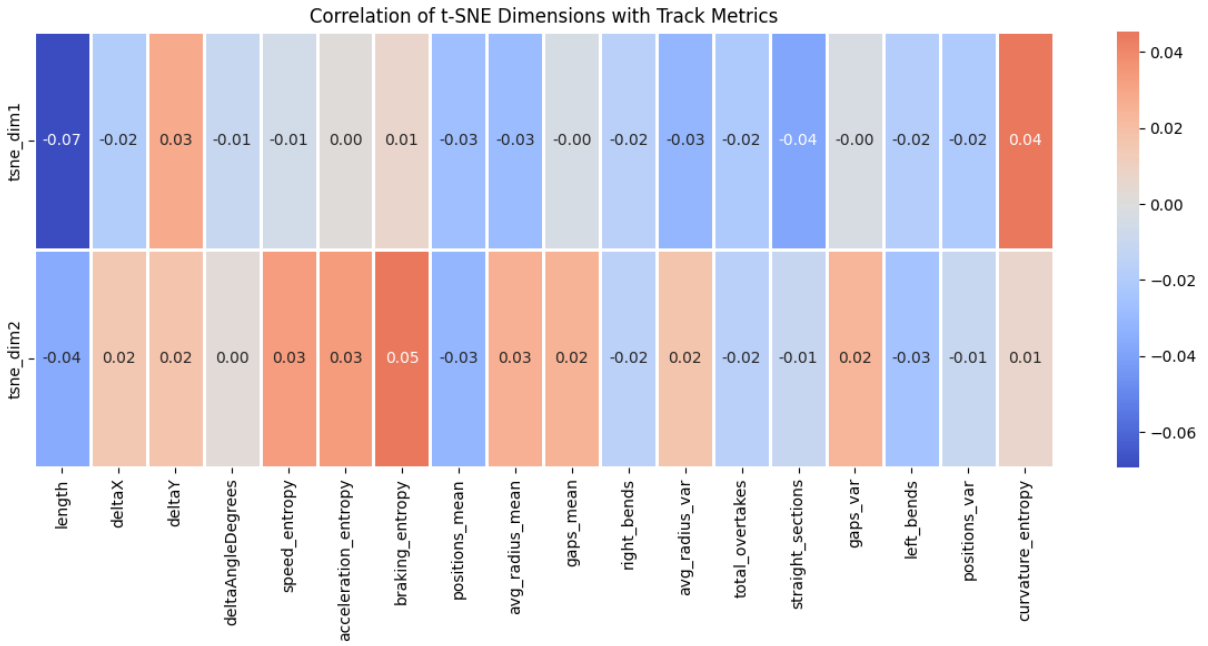

Figure 4.22: Spearman correlations between t-SNE embedding dimensions and track metrics. The weak correlations across both dimensions indicate limited interpretability of the t-SNE space in terms of explicit track features.

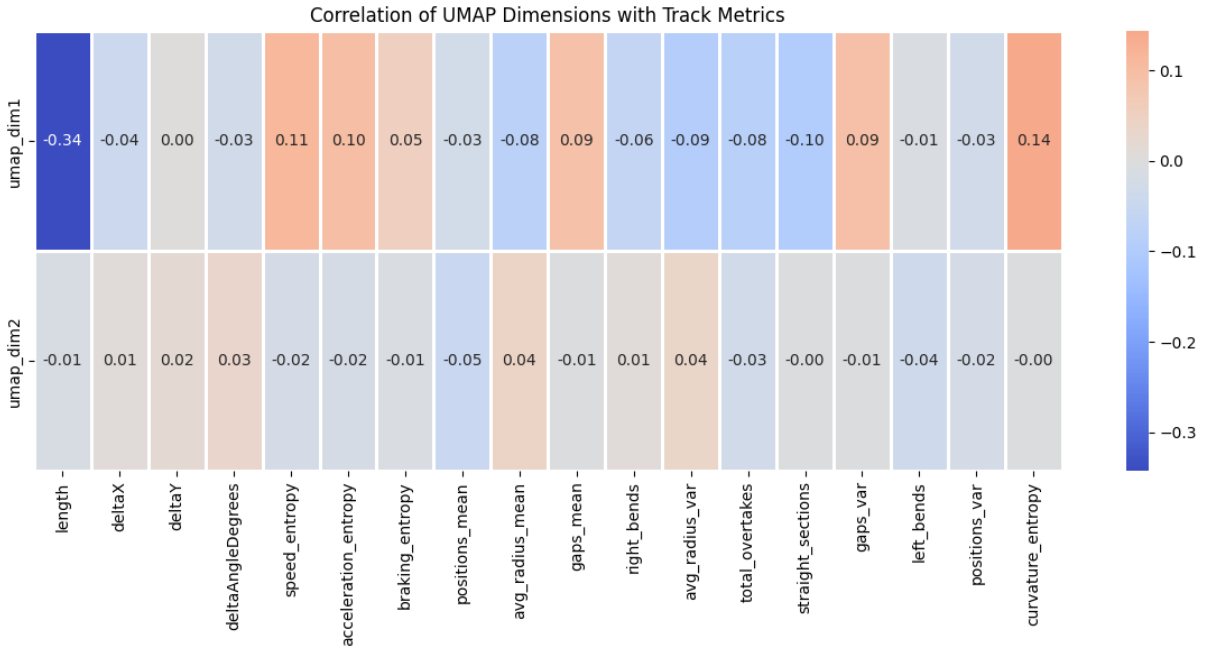

Figure 4.23: Spearman correlations between UMAP embedding dimensions and track metrics. Note the weak signal in Dimension 1 and the lack of any significant correlation in Dimension 2.

This lack of strong, clear correlations reveals a fundamental challenge in using dimensionality reduction for track characterization. While UMAP and t-SNE successfully reveal visual structure in the data, translating this structure into actionable design insights proves difficult. The weak correlations between embedding dimensions and explicit track descriptors suggest that these dimensionality reduction techniques primarily capture high-level, holistic geometric relationships rather than specific, interpretable features.

This disconnect between visual clustering and measurable track properties highlights that while embeddings can effectively group tracks with similar overall shapes, they do not necessarily capture the specific features that define track design quality or gameplay experience. The weak correlations indicate that visual similarity in embedding space does not strongly predict functionally important characteristics such as curvature complexity, overtaking opportunities, or speed dynamics.

Interestingly, we then found that the empty spaces between two clusters can be filled with self-overlapping tracks, which are topologically invalid. This finding suggests that sparse regions in embedding space serve as natural indicators of design quality, with dense clusters representing geometrically sound track configurations and sparse areas harboring problematic layouts.

UMAP for Behavioral Descriptor Generation

Despite this interpretability limitation, the majority of our MAP-Elites experiments utilized UMAP dimensions as behavioral descriptors. This choice was motivated by UMAP's ability to capture complex, non-linear geometric relationships that are difficult to hand-engineer through explicit metrics.

UMAP-based descriptors offer several advantages; they preserve geometric similarity (placing tracks with subtle similarities closer in embedding space), provide dimensionality efficiency through a compact two-dimensional representation, and enable non-linear feature discovery that emerges from the data itself rather than predefined characteristics.

However, this approach sacrifices interpretability. While hand-crafted descriptors like track length or curvature entropy have direct, understandable meanings, UMAP dimensions lack inherent interpretability beyond geometric similarity. This makes it challenging to understand precisely what track features drive the evolutionary search, though the potential for discovering novel design relationships that might be overlooked by conventional metrics justified this trade-off.

4.2. Experiments

This section details the experiments conducted to evaluate the Voronoi and Convex Hull techniques for racing track generation. The objective was to assess the ability of these techniques to produce diverse, high-quality tracks that met the design criteria established in the preliminary analysis. These experiments utilized MAP-Elites with sliding boundaries to examine the impact of different track representations on generated maps. The study sought to evaluate how each representation captures diversity and quality in racing

tracks, and how well it illuminates the feature space.

Based on insights from noisiness, correlation, and clustering analyses, five near-orthogonal descriptors guided these MAP-Elites experiments: length, curvature entropy, gaps mean, normalized *total_ overtakes* (*total_ overtakes* divided by delta), and speed entropy. This selection provides broad behavioral coverage, capturing distinct aspects of track characteristics including scale, curvature variability, overtaking dynamics, vehicle speed dynamics, and gap distribution. This approach also mitigated redundancy and minimized noise-related biases identified during preliminary assessments. These outcomes informed subsequent evolutionary experiments, guiding descriptor performance and robustness in MAP-Elites explorations.

4.2.1. Convex Hull Technique Experiments

We conducted an experiment with a simple formula. Since preliminary analysis suggested that length was highly correlated with UMAP Dimension 1, its inclusion in the fitness function was considered potentially redundant. If the behavioral descriptors already encourage diversity in track size and shape, the fitness function could be simplified to focus purely on a gameplay-centric measure of quality. We therefore tested a configuration with a fitness function consisting only of the normalized *total_ overtakes* score:

$$\text{score} = \frac{total\_\ overtakes}{(delta + 10^{-3})} \tag{4.3}$$

Here, delta represents the sum of positional discrepancies (deltaX + deltaY), penalizing tracks with poor geometric closure. This formulation emphasizes overtaking opportunities while rewarding stable, well-formed tracks. For behavioral descriptors, we selected UMAP Dimension 1 and *speed_ entropy*. This choice was deliberate: UMAP Dimension 1 acts as a data-driven proxy for overall geometric complexity, while *speed_ entropy* was identified as a highly orthogonal feature capturing the dynamic driving experience.

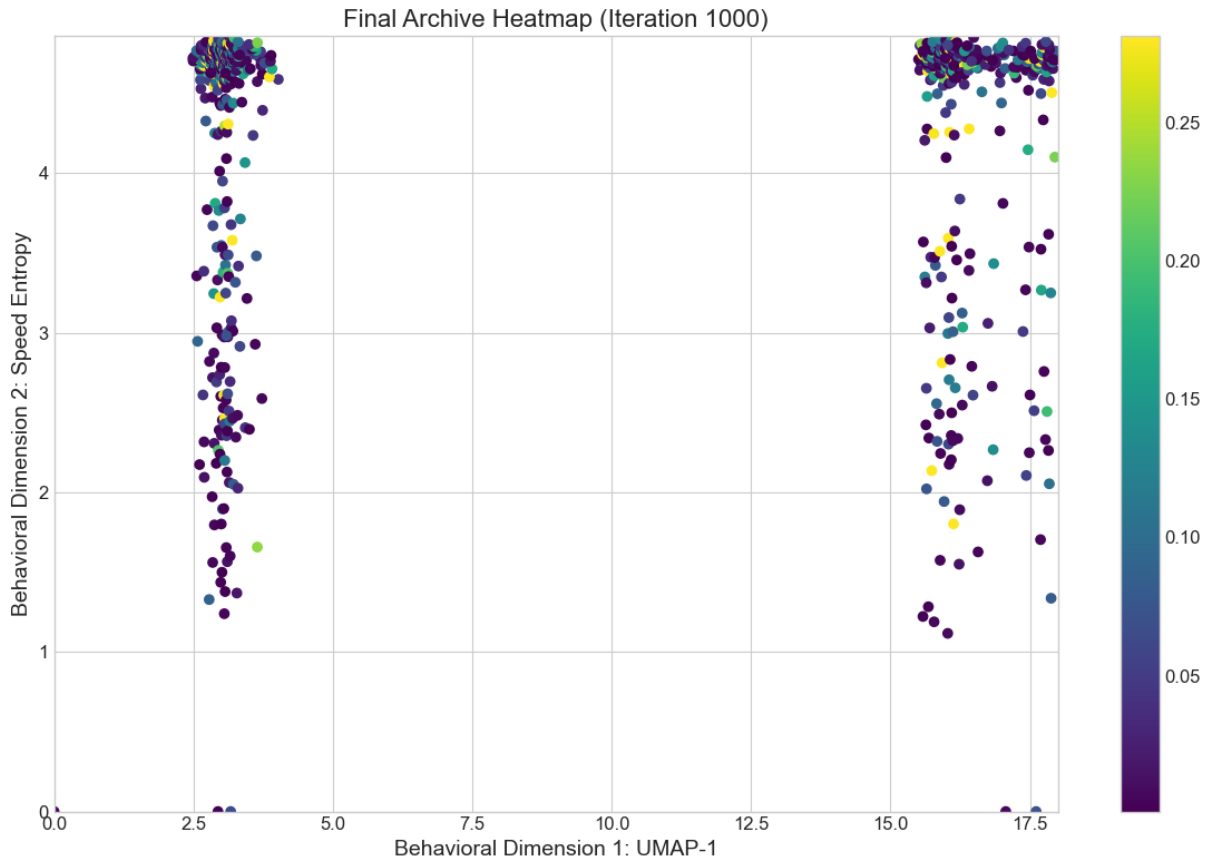

Figure 4.24: MAP-Elites archive using UMAP Dimension 1 and *speed_entropy* as descriptors.

The resulting archive (Figure 4.24) confirmed our hypothesis. A clear pattern emerged where regions of low speed entropy (indicating simpler, less dynamic tracks) struggled to produce solutions with high overtaking scores. This is an expected and logical outcome, as tracks with low variability naturally offer fewer opportunities for pilots.

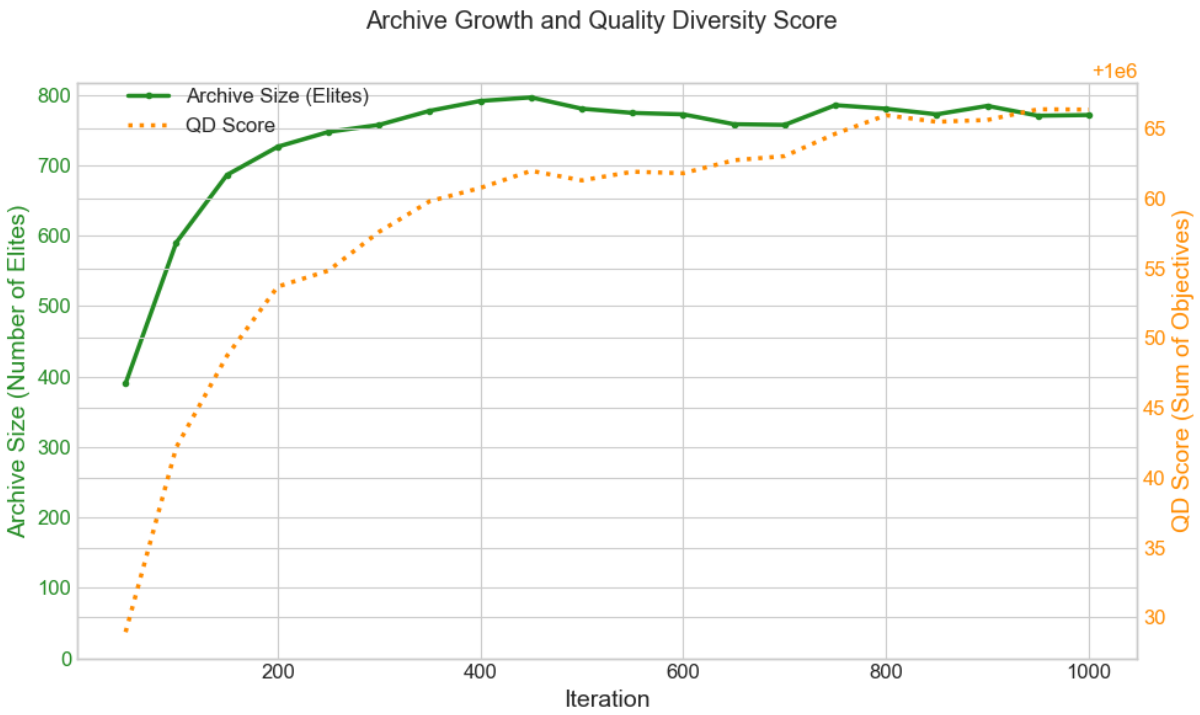

Figure 4.25: QD Score evolution for the Convex Hull experiment over 1000 iterations.

4.2.2. Voronoi Technique Experiments

The initial experiment was then replicated using the Voronoi technique to evaluate its performance under the same conditions and allow for a direct comparison. We used the identical fitness function based on normalized overtakes and the same behavioral descriptors (UMAP Dimension 1 and speed entropy):

$$\text{score} = \frac{total\_overtakes}{(delta + 10^{-3})} \tag{4.4}$$

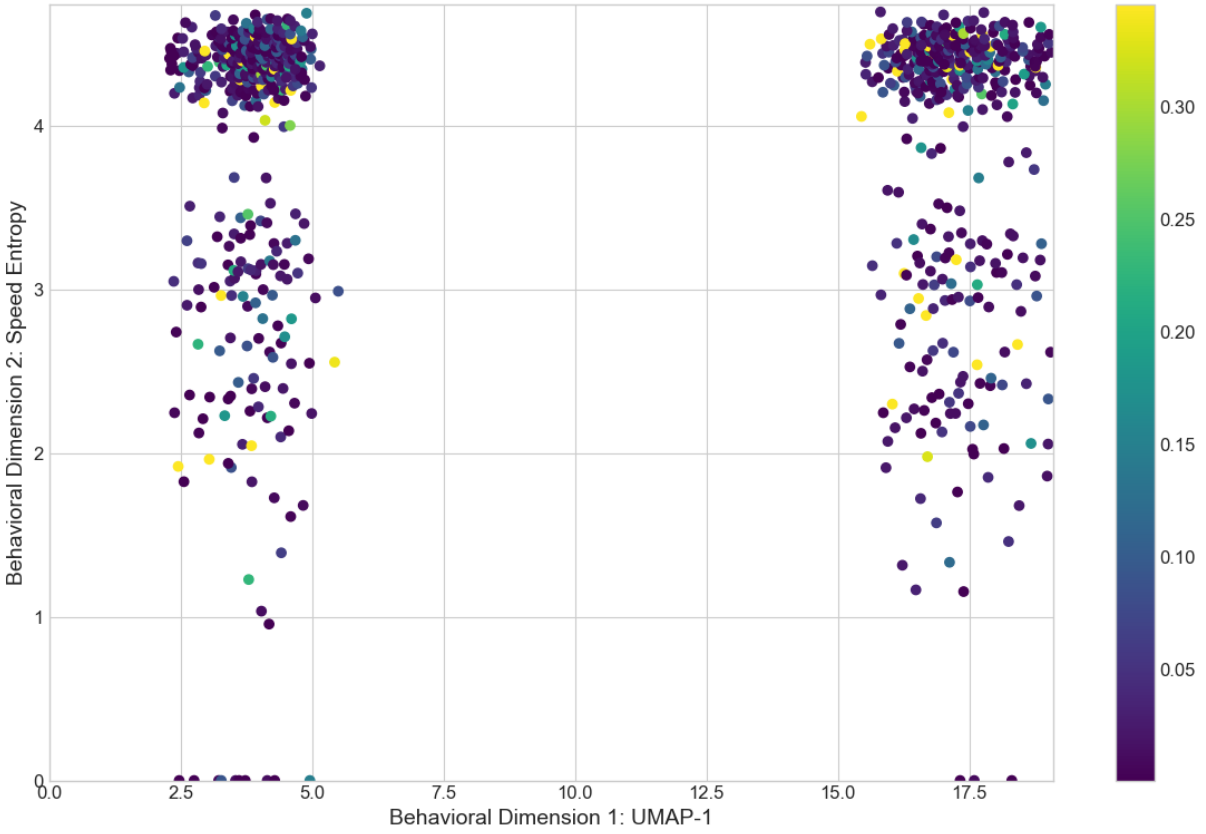

Figure 4.26: MAP-Elites archive for the Voronoi technique, using UMAP Dimension 1 and *speed_ entropy* as descriptors.

Figure 4.27: QD Score evolution for the Voronoi experiment over 1000 iterations.

As shown in Figure 4.26, the resulting archive for the Voronoi technique strongly corroborated the results of the Convex Hull experiment. It likewise revealed that tracks characterized by low speed entropy were unable to achieve high fitness scores. This consistent outcome across both generative representations provides robust evidence that a track's dynamic variability is fundamentally linked to its potential for facilitating overtaking maneuvers, validating our choice of *speed_entropy* as an effective behavioral descriptor.

Main Experiment and Operator Analysis

This foundational understanding confirmed our decision to proceed with a more comprehensive setup for the main experiments. We tried a more elaborated fitness function that incorporates multiple behavioral descriptors, allowing for a richer exploration of the design space. The fitness function was defined as follows:

$$\text{score} = \text{length} + \text{right\_bends} + \frac{\text{total\_overtakes}}{(\text{delta} + 10^{-3})} \tag{4.5}$$

Within this setup, a key question was to understand the individual contributions of our novel crossover operators. To investigate this, we conducted a comparative analysis between the two primary methods: the Random-Line Partitioning Method and the Relative Reconstruction Method. This allowed us to isolate their effects and quantify their respective strengths in exploring the design space.

Figure 4.28: Evolution of Archive Size and QD Score for Random-Line Partitioning crossover operator over 1000 iterations.

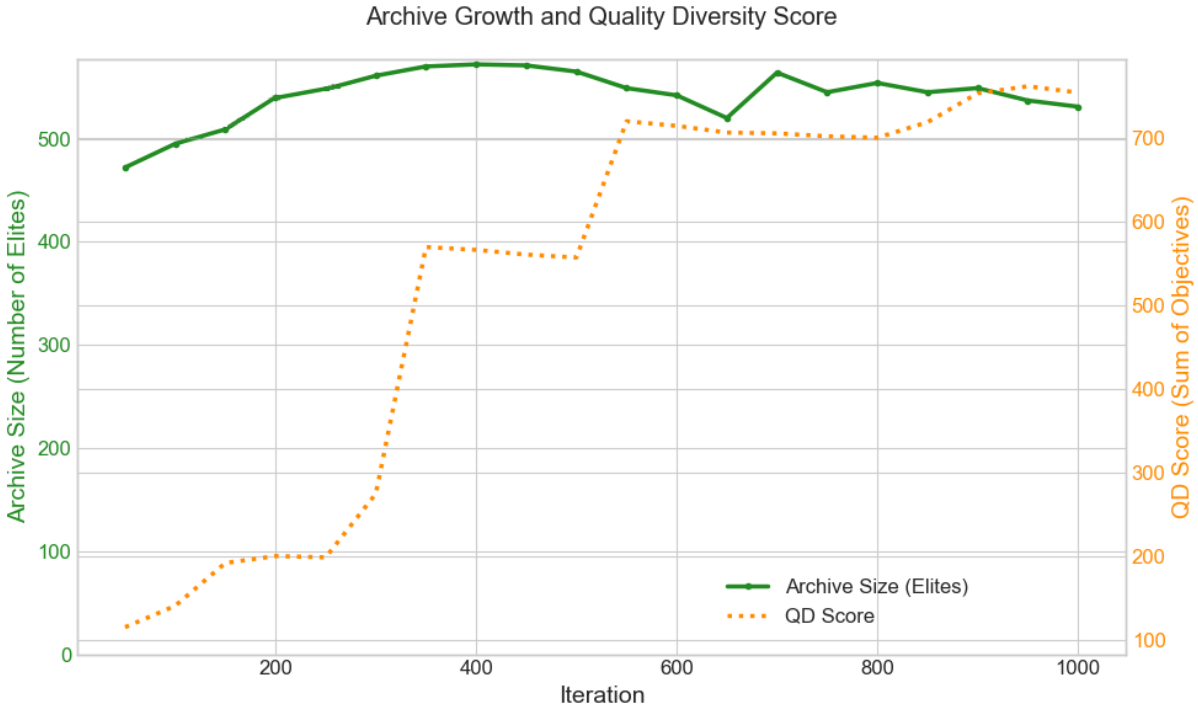

Figure 4.29: Evolution of Archive Size and QD Score for Relative Mapping crossover operator over 1000 iterations.

The comparison revealed that the Relative Reconstruction Method (Figure 4.29) achieved better results compared to the Random-Line Partitioning Method (Figure 4.28).

Final Archive and Qualitative Analysis

Overall, the Voronoi method demonstrated superior efficiency in archive coverage and diversity compared to the Convex Hull approach, an advantage anticipated from its greater geometric expressiveness. The final archive is shown in Figure 4.30. This run achieved a size of 532 elites with 59.11% coverage of the behavioral space.

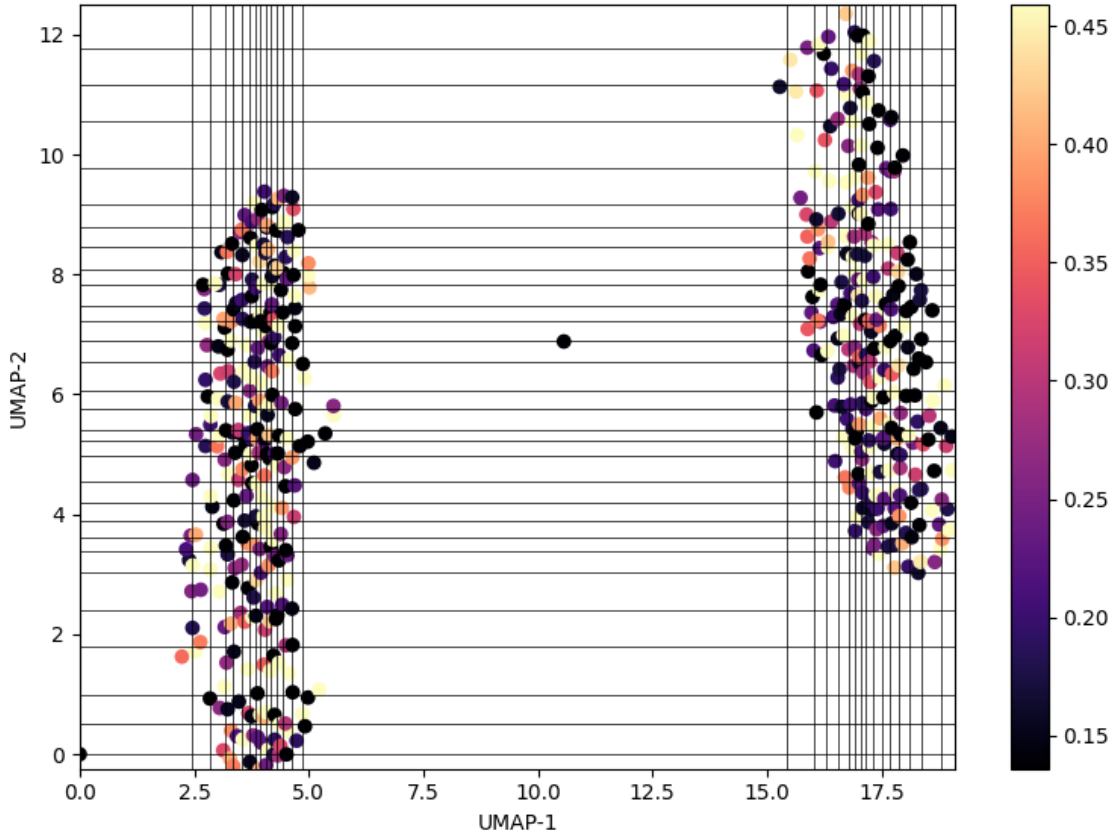

Figure 4.30: Final MAP-Elites archive scatterplot for Voronoi tracks using the Relative Reconstruction Method at 1000 iterations.

The following collection of high-fitness tracks from this archive demonstrates the diversity and quality achieved.

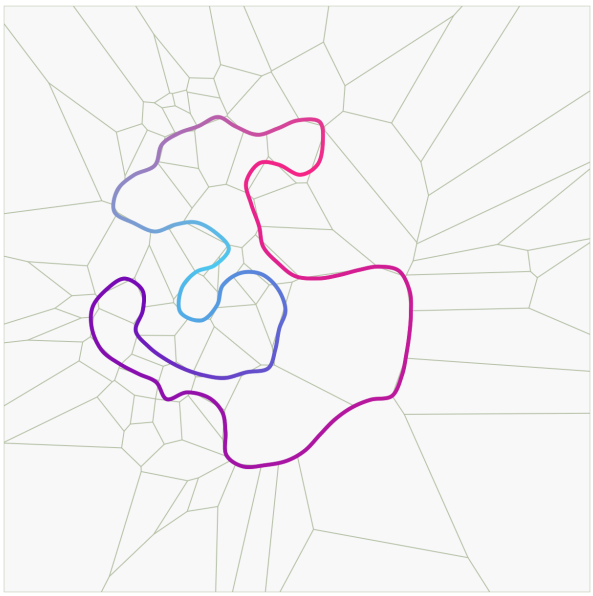

Figure 4.31: Example high-fitness track (ID: 984.11). Key metrics: Length=1509.22m, Overtakes=28, Curvature Entropy=3.65, Speed Entropy=2.89.

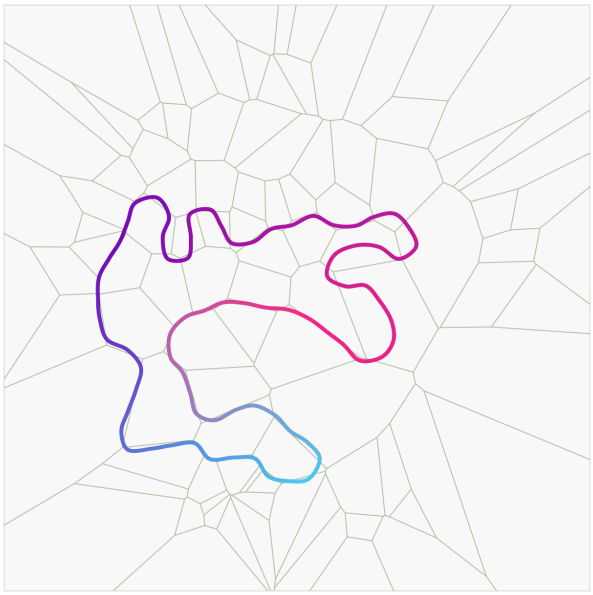

Figure 4.32: Example high-fitness track (ID: 985.11). Key metrics: Length=1698.80m, Overtakes=35, Curvature Entropy=3.78, Speed Entropy=2.73.

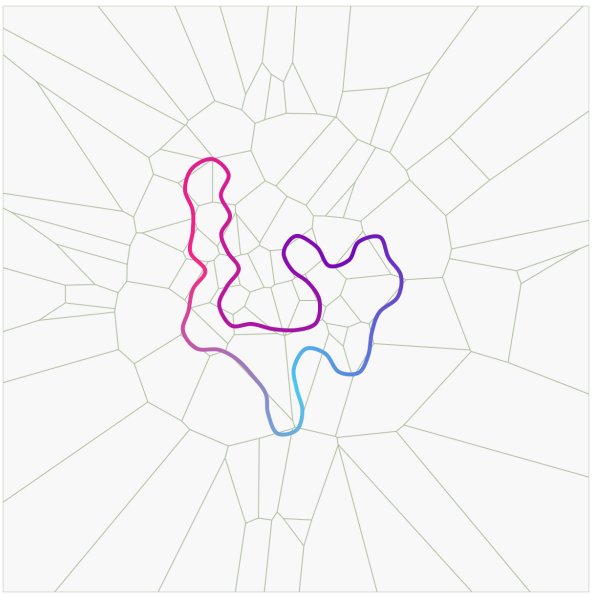

Figure 4.33: Example high-fitness track (ID: 983.77). Key metrics: Length=1210.69m, Overtakes=25, Curvature Entropy=3.94, Speed Entropy=4.46.

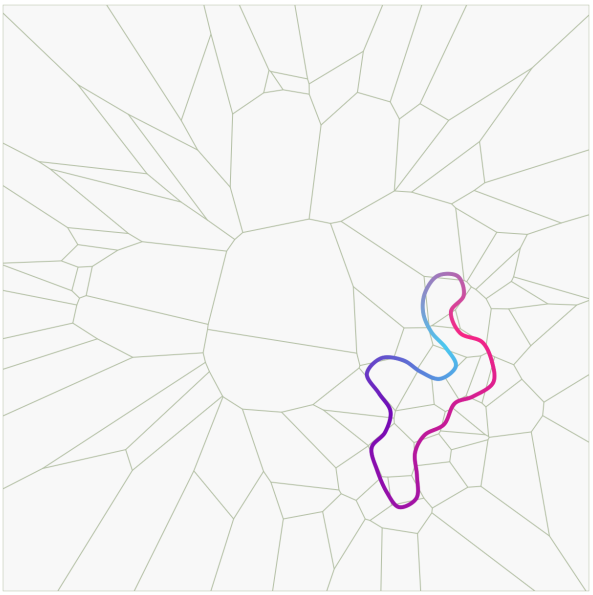

Figure 4.34: Example high-fitness track (ID: 956.95). Key metrics: Length=716.09m, Overtakes=2, Curvature Entropy=4.04, Speed Entropy=3.19.

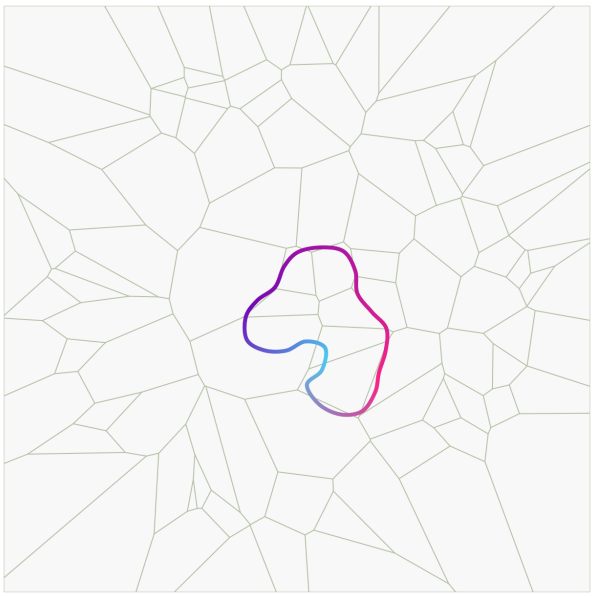

Figure 4.35: Example high-fitness track (ID: 995.44). Key metrics: Length=547.69m, Overtakes=14, Curvature Entropy=3.32, Speed Entropy=4.46.

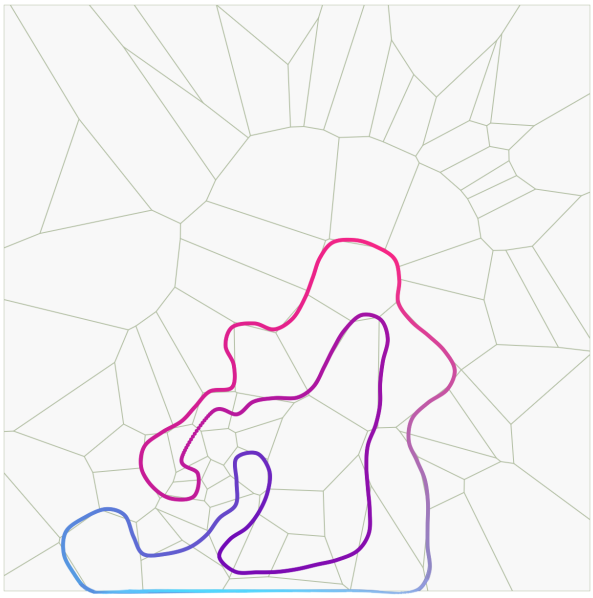

Figure 4.36: Example high-fitness track (ID: 973.63). Key metrics: Length=2447.02m, Overtakes=61, Curvature Entropy=3.35, Speed Entropy=4.21.

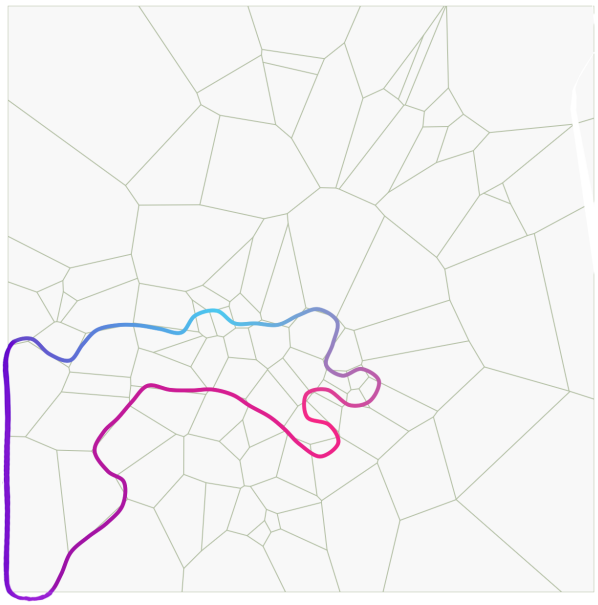

Figure 4.37: Example high-fitness track (ID: 963.57). Key metrics: Length=1461.58m, Overtakes=5, Curvature Entropy=3.87, Speed Entropy=1.81.

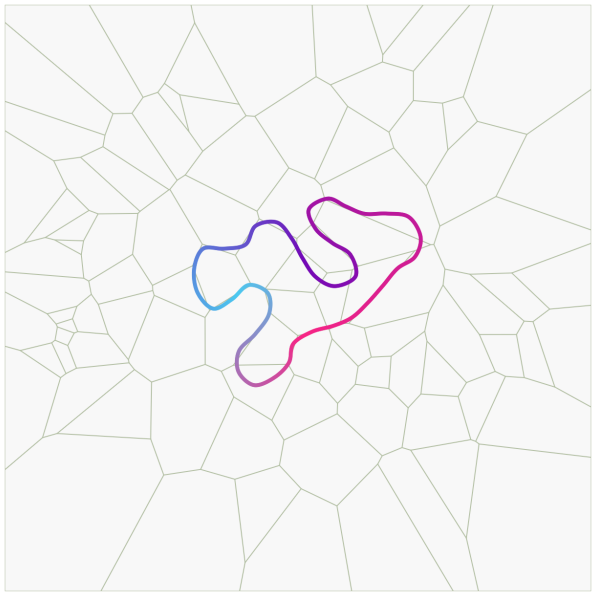

Figure 4.38: Example high-fitness track (ID: 957.58). Key metrics: Length=924.49m, Overtakes=1, Curvature Entropy=3.83, Speed Entropy=4.03.

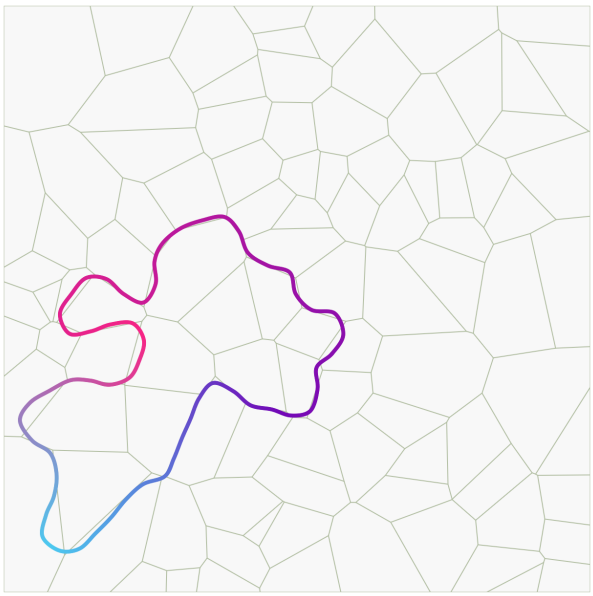

Figure 4.39: Example high-fitness track (ID: 504.22). Key metrics: Length=1330.63m, Overtakes=366, Curvature Entropy=3.76, Speed Entropy=4.03.

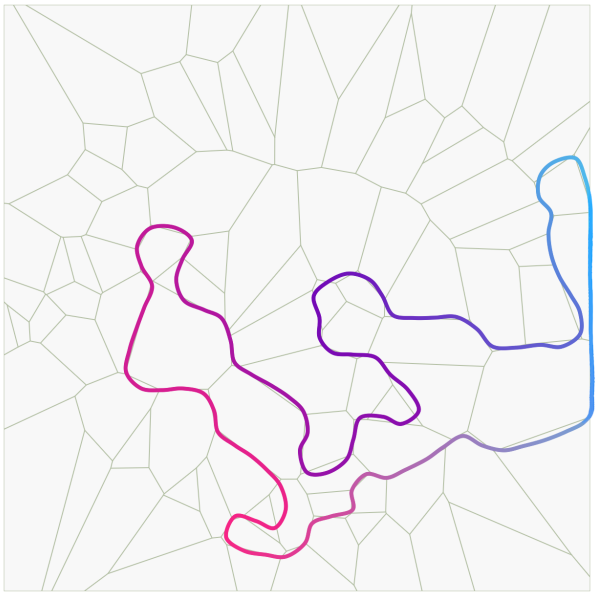

Figure 4.40: Example high-fitness track (ID: 970.69). Key metrics: Length=2451.16m, Overtakes=5, Curvature Entropy=3.37, Speed Entropy=2.08.

A notable discovery emerged from inspecting this final archive. A track with a high fitness score was found to be topologically invalid, exhibiting self-intersection (Figure 4.41). This observation validates the insight from our UMAP analysis: empty or sparsely populated regions in the latent space often correspond to geometrically problematic designs. Because the TORCS engine does not rigorously validate all topological properties, such

flawed tracks can complete simulations and receive high fitness scores, creating misleading evaluations. This finding underscores the importance of integrating robust geometric checks into the genotype-to-phenotype mapping process to ensure the search algorithm focuses its efforts on valid and meaningful regions of the design space.

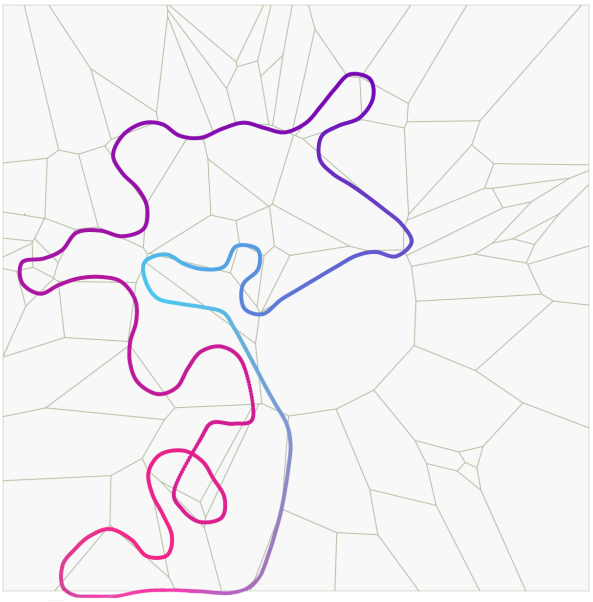

Figure 4.41: Track (ID: 937.93) exhibiting self-intersection. While the simulation completed without errors, this track represents a topological invalidity that highlights limitations in current validation mechanisms.

5 | Conclusions and Future Developments

In this thesis, we successfully demonstrated that MAP-Elites is a powerful tool for the procedural generation of racing tracks. Using the TORCS open-source racing game, we developed a system that leverages evaluation functions computed from simulated racing competitions to guide the evolutionary search towards a diverse archive of high-quality tracks.

We designed and implemented an end-to-end pipeline that seamlessly integrates a Node.js backend for track generation, Docker-containerized TORCS instances for parallel simulation, and a Python-based evolutionary engine powered by the Pyribs library. We investigated two distinct genotype representations—Convex Hulls and a novel method based on Voronoi diagrams—and developed tailored crossover and mutation operators for each. Furthermore, we explored the feature space of competitive racing tracks, defining relevant measures to characterize their geometric and gameplay properties.

A cornerstone of this work was a rigorous preliminary analysis to identify and mitigate the inherent noisiness of emergent gameplay metrics. These initial experiments revealed the high sensitivity of certain metrics, e.g., *total_ overtakes*, to minor artifacts like positional discrepancies at the start/finish line. This critical insight led to the development of a normalized overtaking score that proved far more reliable for evaluating track quality. This data-centric approach to defining behavioral descriptors and the fitness function was crucial for guiding the evolutionary search effectively. Finally, we pioneered the use of dimensionality reduction techniques, specifically UMAP, to automatically generate behavioral descriptors from raw track spline data. A key observation was that sparse or empty regions in the resulting latent space often corresponded to topologically invalid tracks, providing a valuable insight into the structure of the design space.

While the current research provides a robust framework, it is important to acknowledge its limitations, which in turn illuminate clear directions for future work.

A primary limitation stems from the track representation within TORCS and Speed Dreams itself. The XML format, which encodes a track as a sequence of segments, is straightforward but lacks support for crucial features like elevation changes and banked corners. This inherently restricts the generated content to flat, two-dimensional layouts. Furthermore, this sequential definition makes it difficult to guarantee perfect track closure. This often results in a small but measurable "delta error"—a gap between the start and end points—which, as our analysis showed, can introduce noise and corrupt the evaluation

of gameplay metrics.

The reliance on TORCS's built-in AI for gameplay simulation also presents a notable gap between automated evaluation and human experience. While necessary for evaluating thousands of tracks automatically, these bots cannot fully replicate the nuanced, adaptive, and often unpredictable behavior of human players. Consequently, a track that is "fit" for an AI might not be optimally enjoyable for a human. Future work could integrate human evaluation within a co-creative setup, allowing players to provide direct feedback and guide the evolutionary search more effectively.

Finally, the definition of "fitness" itself is context-dependent and represents a significant challenge. The fitness function developed in this thesis is tailored for competitive, simulation-style racing, prioritizing metrics like overtaking opportunities. However, gameplay objectives can vary dramatically across different genres. In a combat racing game, for instance, a "good" track might be one that facilitates strategic use of weapons, with well-placed power-ups, choke points, and areas of cover. The fitness function for such a game would be entirely different, highlighting that a truly generalist track generator must be able to adapt its definition of quality to the specific gameplay experience it aims to create.

Directions for future research emerge:

- **Overcoming Framework Limitations by Migrating to a Modern Engine:** A crucial next step is to overcome the inherent limitations of the TORCS framework by migrating the pipeline to a modern game engine. Such engines would resolve the geometric and topological issues identified in this work—including self-intersections and track closure errors—through robust, native 3D geometry handling, eliminating the need for algorithmic workarounds. More importantly, this transition would dramatically expand the creative design space. It would enable the procedural generation of true 3D tracks with complex elevation changes and banked corners, as well as the surrounding environment (terrain, barriers), leading to far more realistic and immersive experiences. This represents a fundamental shift from patching the constraints of an older system to fully leveraging the power of modern procedural content generation.
- **Enhancing Evaluation with Human-in-the-Loop:** To better align generated content with human enjoyment, an interactive evolution approach could be adopted. Such a system would allow a human player to rate tracks or specific segments, integrating this subjective feedback directly into the fitness score to guide the algorithm toward more engaging designs.
- **Alleviating the Simulation Bottleneck:** To improve computational efficiency, surrogate models (e.g., trained neural networks) could be developed. These models could learn to predict a track's fitness score based on its genotype or simple geometric features, providing a rapid evaluation for most candidates and reserving full simulation for only the most promising individuals.
- **Exploring Advanced QD Algorithms:** The adoption of cutting-edge algorithms like Descriptor-Conditioned Gradients (DCG-MAP-Elites) could yield significant ef-

ficiency gains. If the track generation process can be made differentiable, gradients could guide the search towards promising regions of the design space far more directly than is possible with mutation alone.

In conclusion, this thesis has successfully laid the groundwork for using Quality Diversity to automate and enhance the creative process of racing track design. The developed pipeline and the insights gained from our experiments provide a solid foundation upon which future research can build, pushing the boundaries of what is possible in procedural content generation and paving the way for endlessly diverse, engaging, and personalized gaming experiences.

Bibliography

[1] Bernhard Wymann, Christos Dimitrakakis, Andrew Sumner, Eric Espie, and Christophe Guionneau. TORCS: The open racing car simulator, 2001. URL `http://torcs.sourceforge.net/`.

[2] Speed Dreams Development Team. Speed dreams, 2010. URL `http://speed-dreams.sourceforge.net/`.

[3] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, April 2015. doi: 10.48550/arXiv.1504.04909.

[4] Atari, Inc. Space race. `https://en.wikipedia.org/wiki/Space_Race_(video_game)`, July 1973.

[5] Namco. Pole position. `https://en.wikipedia.org/wiki/Pole_Position`, September 1982.

[6] Sega. Out run. `https://en.wikipedia.org/wiki/Out_Run`, September 1986.

[7] Nintendo EAD. F-zero. `https://en.wikipedia.org/wiki/F-Zero_(video_game)`, November 1990.

[8] Sega AM2. Virtua racing. `https://en.wikipedia.org/wiki/Virtua_Racing`, August 1992.

[9] Nintendo EAD. Super mario kart. `https://en.wikipedia.org/wiki/Super_Mario_Kart`, August 1992.

[10] Polyphony Digital. Gran turismo. `https://en.wikipedia.org/wiki/Gran_Turismo`, December 1997.

[11] Sony Interactive Entertainment. Gran turismo 7. `https://en.wikipedia.org/wiki/Gran_Turismo_7`, March 2022.

[12] Kunos Simulazioni. Assetto corsa. `https://en.wikipedia.org/wiki/Assetto_Corsa`, December 2014.

[13] KUNOS Simulazioni. Assetto corsa evo on steam. `https://store.steampowered.com/app/3058630/Assetto_Corsa_EVO/`, 2025.

[14] Milestone s.r.l. Ride. `https://en.wikipedia.org/wiki/Ride_(video_game)`, March 2015.

[15] Playground Games. Forza horizon. `https://en.wikipedia.org/wiki/Forza_Horizon`, October 2012.

[16] Playground Games & Xbox Game Studios. Forza horizon 5. https://en.wikipedia.org/wiki/Forza_Horizon_5, November 2021.

[17] Nadeo. Trackmania. https://en.wikipedia.org/wiki/TrackMania_(2003_video_game), November 2003.

[18] Daniel Michelon De Carli, Fernando Bevilacqua, Cesar Tadeu Pozzer, and Marcos Cordeiro d'Ornellas. A Survey of Procedural Content Generation Techniques Suitable to Game Development. In *2011 Brazilian Symposium on Games and Digital Entertainment*, pages 26–35, Salvador, November 2011. IEEE. ISBN 978-1-4673-0797-0 978-0-7695-4648-3. doi: 10.1109/SBGAMES.2011.15.

[19] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011. ISSN 1943-068X, 1943-0698. doi: 10.1109/TCIAIG.2011.2148116.

[20] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. Search-based procedural content generation. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Anikó Ekárt, Anna I. Esparcia-Alcázar, Chi-Keong Goh, Juan J. Merelo, Ferrante Neri, and Mike Preuß, editors, *Applications of Evolutionary Computation*, volume 6024, pages 141–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-12238-5 978-3-642-12239-2. doi: 10.1007/978-3-642-12239-2_15.

[21] Jiao Jian Wang and Olana Missura. Racing tracks improvisation. In *2014 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, Dortmund, Germany, 2014. IEEE. ISBN 978-1-4799-3547-5. doi: 10.1109/CIG.2014.6932899.

[22] Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Automatic track generation for high-end racing games using evolutionary computation. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):245–259, 2011. ISSN 1943-068X, 1943-0698. doi: 10.1109/TCIAIG.2011.2148116.

[23] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*, pages 395–402, Dublin, Ireland, 2011. ACM. ISBN 978-1-4503-0557-0. doi: 10.1145/2001576.2001634.

[24] Jonas Freiknecht. *Procedural Content Generation for Games*. Doctoral dissertation, University of Mannheim, Mannheim, Germany, 2021. URL https://madoc.bib.uni-mannheim.de/59000/. Supervisor: Wolfgang Effelsberg; available via University of Mannheim repository.

[25] Gustavo Maciel. Generating procedural racetracks. Gamasutra Blog,

December 2013. URL https://www.gamedeveloper.com/programming/generating-procedural-racetracks.

[26] Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, September 1991. ISSN 0360-0300, 1557-7341. doi: 10.1145/116873.116880.

[27] Yoav I. H. Parish and Pascal M"uller. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*, pages 301–308. ACM Press, 2001. doi: 10.1145/383259.383292.

[28] S. A. Groenewegen, R. M. Smelik, K. J. de Kraker, and R. Bidarra. Procedural city layout generation based on urban land use models. In *Eurographics Workshop on Procedural Content Generation in Games*, pages 31–38, 2009. doi: 10.2312/EGPCG/EGPCG09/031-038.

[29] Markus Lipp, Daniel Scherzer, Peter Wonka, and Michael Wimmer. Interactive modeling of city layouts using layers of procedural content. *Computer Graphics Forum*, 30(2):345–354, 2011. doi: 10.1111/j.1467-8659.2011.01865.x.

[30] Luca Garattoni. Procedural engine for sim racing track generation: The basic road-map. LinkedIn Article, July 2015. URL https://www.linkedin.com/pulse/procedural-engine-sim-racing-track-generation-basic-luca-garattoni.

[31] Sean Darwiche and Martin Nystr"om. Finding junctions in spline-based road generation. Technical Report DivA-53417, Malm"o University, Faculty of Technology and Society, 2022. URL https://www.diva-portal.org/smash/record.jsf?pid=diva2:1675311.

[32] R. M. Smelik, R. Bidarra, E. Kruijff, and K. J. de Kraker. A survey of procedural methods for terrain modelling. *International Journal of Virtual Reality*, 9(2):1–20, 2009.

[33] Marco Tanzola. Racing Tracks Generation via Generative Adversarial Networks. Master's thesis, Politecnico di Milano, 2022. URL https://hdl.handle.net/10589/210856.

[34] Justin K. Pugh, Louis B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016. doi: 10.3389/frobt.2016.00040.

[35] Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011. doi: 10.1162/EVCO_a_00025.

[36] Joel Lehman and Kenneth O. Stanley. Evolving a diversity of creatures through novelty search and local competition. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO '11)*, pages 211–218, 2011. doi: 10.1145/2001576.2001606.

[37] Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Using centroidal voronoi tessellations to scale up the multi-dimensional archive of phe-

notypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22(4): 623–630, 2018. doi: 10.1109/TEVC.2018.2797052.

[38] Matthew C. Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K. Hoover. Covariance matrix adaptation for the rapid illumination of behaviour space. *Nature Machine Intelligence*, 2:361–369, 2020. doi: 10.1038/s42256-020-0181-2.

[39] Antoine Cully. Multi-emitter map-elites: Improving quality, diversity and convergence speed with heterogeneous sets of emitters. *Evolutionary Computation*, 29(2):263–289, 2021. doi: 10.1162/evco_a_00292. Originally published as arXiv:2007.05352 (2020).

[40] Nicolas Pierrot, Pierre Flageat, and Jeff Clune. Qd-pg and me-ls: Policy-gradient and low-spread variants of map-elites. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '23)*, pages 1234–1242, 2023. doi: 10.1145/3583131.3590399.

[41] Daniele Gravina, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. Procedural content generation through quality diversity. In *Proceedings of the IEEE Conference on Games (CoG)*, pages 1–8, London, UK, 2019. doi: 10.1109/CIG.2019.8848053.

[42] Maxence Faldor, Félix Chalumeau, Manon Flageat, and Antoine Cully. MAP-Elites with descriptor-conditioned gradients and archive distillation into a single policy. arXiv preprint arXiv:2303.03832, 2023.

[43] Bryon Tjanaka, Matthew C Fontaine, David H Lee, Yulun Zhang, Nivedit Reddy Balam, Nathaniel Dennler, Sujay S Garlanka, Nikitas Dimitri Klapsis, and Stefanos Nikolaidis. Pyribs: A bare-bones python library for quality diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '23, page 220–229, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701191. doi: 10.1145/3583131.3590374. URL `https://doi.org/10.1145/3583131.3590374`.

[44] Lauren McCarthy and contributors. p5.js. A JavaScript library for creative coding, based on the core principles of Processing., 2014. Available at `https://p5js.org`.

[45] Node.js Foundation. Node.js. A JavaScript runtime environment built on Chrome's V8 JavaScript engine, 2009. Available at `https://nodejs.org`.

[46] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando P'erez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B. Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Dami'an Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016. doi: 10.3233/978-1-61499-649-1-87.

[47] Docker, Inc. Docker. An open platform for developing, shipping, and running applications in containers., 2013. Available at `https://www.docker.com`.

[48] Jacopo Sirianni. Supporto alla progettazione e analisi di tracciati per un simulatore di guida. Tesi di laurea magistrale (master's thesis), Politecnico di Milano, Milan, Italy, jul 2016. URL `https://hdl.handle.net/10589/122987`. Supervisor: Daniele Loiacono.

A | Source Code

This thesis is supported by a full implementation of the procedural content generation pipeline. The complete source code, including the MAP-Elites implementation, track generation algorithms, Docker configurations, and telemetry analysis tools, is publicly available at the following GitHub repository:

https://github.com/martinopiaggi/Quality-Diversity-for-Racing-Track-Design

Furthermore, the web-based visualization tool, which was instrumental for the development and debugging of the genetic operators (as shown in Figures 3.6, 3.7, and 3.8), is deployed as a live web application. It allows for hands-on exploration of the track generation process and can be accessed at:

https://pcgtrack.netlify.app

Acknowledgements

I would like to sincerely thank Prof. Lanzi and Prof. Loiacono for giving me the chance to end my academic career working on my greatest passion – a path that's now led me to work on racing games in Milestone as a gameplay developer – and for teaching what I consider the best course of my academic career: Videogame Design and Programming.