KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY

Course No: CSE 4110
Course Title: Artificial Intelligence Laboratory

Road to Intelligence: The Agent's Race

Submitted By

Sourav Debnath
Roll No: 2007109

Tirtho Mondal
Roll No: 2007117

Department of Computer Science and Engineering
Khulna University of Engineering & Technology
Khulna–9203, Bangladesh

October, 2025

Road to Intelligence: The Agent's Race

Submitted By

Sourav Debnath
Roll No: 2007109

Tirtho Mondal
Roll No: 2007117

Submitted To

Md. Mehrab Hossain Opi
Lecturer, Department of CSE, KUET

Waliul Islam Sumon
Lecturer, Department of CSE, KUET

Department of Computer Science and Engineering
Khulna University of Engineering & Technology
Khulna 9203, Bangladesh

Contents

List of Figures

List of Tables

Abstract

Autonomous vehicle navigation and real-time decision-making under dynamic constraints are key challenges in intelligent transportation systems. This project presents an Intelligent Bus Race Simulation, where two AI-controlled buses compete in a racing environment filled with dynamic obstacles, pedestrians, and opponent interactions. The system employs a Hybrid AI Architecture combining Breadth-First Search (BFS) Depth-First Search (DFS), Minimax with Alpha-Beta Pruning, and Fuzzy Logic to enable intelligent, adaptive, and competitive decision-making. The simulation is implemented using Python and Pygame, providing a real-time interactive racing environment. The AI agents dynamically select safe lanes using pathfinding algorithms, anticipate opponent moves using adversarial search, and apply fuzzy reasoning to adjust speed near road boundaries. Experimental results demonstrate that the hybrid approach outperforms single-algorithm strategies in terms of survival time, score accumulation, and collision avoidance under varying hazard densities.

1 Introduction

1.1 Motivation and Problem Context

The development of intelligent transportation systems (ITS) has become increasingly critical as vehicular traffic grows in complexity and volume. Modern vehicles require autonomous navigation capabilities that can handle dynamic obstacles, competitive interactions, and uncertain environments in real time. Traditional navigation systems often rely on predefined routes and fail to adapt to rapidly changing conditions or adversarial scenarios.Racing games and competitive autonomous navigation scenarios provide an ideal testbed for developing and evaluating multi-strategy AI systems. Unlike conventional pathfinding problems, racing environments introduce time-critical decision-making, opponent modeling, collision avoidance under speed constraints, and continuous adaptation to hazard distributions. These characteristics make racing simulations highly representative of real-world autonomous driving challenges.

1.2 Aim of the Project

This project aims to design, implement, and evaluate a Hybrid AI Bus Racing System that integrates multiple search and reasoning algorithms to enable intelligent, competitive, and adaptive behavior. The system combines classical AI pathfinding techniques with adversarial game theory and fuzzy control to achieve robust performance under dynamic conditions.

1.3 Objectives

1. To implement a real-time racing simulation environment with dynamic obstacle spawning and lane-based navigation.
2. To design a hybrid AI decision architecture combining BFS, DFS, Minimax, and Fuzzy Logic.
3. To evaluate the performance of hybrid AI against single-algorithm baselines.
4. To analyze collision avoidance, score accumulation, and survival rates under varying hazard densities.
5. To develop an interactive simulation framework using Python and Pygame.

1.4 Scope and Significance

Unlike pure reinforcement learning approaches that require extensive training data, this hybrid architecture leverages classical AI techniques that provide immediate, interpretable decision-making. The modular design allows easy integration of additional algorithms or adaptation to different racing scenarios. The framework can be extended to:

1. Autonomous vehicle lane-change assistance systems
2. Multi-agent coordination in traffic networks
3. Educational simulations for AI and intelligent systems courses
4. Testing grounds for hybrid AI architectures

2 Background Study

2.1 Search Algorithms for Pathfinding

2.1.1 Breadth-First Search (BFS)

Breadth-First Search (BFS) is a systematic graph traversal algorithm that explores all neighboring nodes at a given depth before advancing to nodes at the next level. In lane navigation tasks, BFS is particularly effective because it always finds a solution if one exists, guaranteeing completeness. Moreover, for unweighted graphs such as typical lane navigation scenarios BFS is optimal, ensuring the identification of the shortest possible path to a safe lane. However, these benefits come at the cost of computational resources: both its time complexity and space complexity scale exponentially, given by $O(b^d)$, where b represents the branching factor and d denotes the maximum depth of the search.

2.1.2 Depth-First Search (DFS)

Depth-First Search (DFS) is a graph traversal algorithm that explores as far as possible along each branch before backtracking. In the context of racing scenarios with limited lookahead depth, DFS can rapidly find feasible lane-change sequences without needing to exhaustively explore all possibilities. It is particularly space efficient, with a space complexity of $O(bd)$, where b is the branching factor and d is the maximum depth. DFS can quickly reach deep solutions, making it useful for fast decision-making. However, it is not guaranteed to find the shortest path and may sometimes miss optimal solutions. To manage computation, DFS can also be depth-limited to restrict the extent of exploration.

2.2 Adversarial Search

2.2.1 Minimax Algorithm

Minimax is a decision-making algorithm used in two-player games where one player aims to maximize their advantage while the opponent aims to minimize it. In racing, this models the competitive interaction between buses.The algorithm recursively evaluates game states by alternating between maximizing and minimizing players:

$$\text{minimax}(s, d, \text{maximizing}) = \begin{cases} \text{eval}(s), & \text{if } d = 0 \\ \max\limits_{s' \in \text{children}(s)} \text{minimax}(s', d-1, \text{false}), & \text{if maximizing} \\ \min\limits_{s' \in \text{children}(s)} \text{minimax}(s', d-1, \text{true}), & \text{otherwise} \end{cases} \quad (1)$$

2.2.2 Alpha-Beta Pruning

Alpha-beta pruning is an optimization technique for the minimax algorithm that eliminates branches in the game tree which cannot possibly affect the final decision. It operates by maintaining two critical values during search:

1. α: The best value currently achievable by the maximizing player.
2. β: The best value currently achievable by the minimizing player.

This process enables more efficient evaluations by reducing the number of nodes the algorithm needs to examine, thereby speeding up decision-making without sacrificing accuracy. Pruning occurs when $\beta \leq \alpha$, significantly reducing the search space without affecting optimality.

2.3 Fuzzy Logic Control

Fuzzy logic provides a mathematical framework for reasoning with imprecise or uncertain information. Unlike binary logic, fuzzy logic allows partial membership in sets, making it ideal for continuous control problems.Fuzzy Logic Components:Fuzzy logic in this project comprises three main components: fuzzification, which converts crisp input values to fuzzy membership values; inference, where fuzzy rules are applied to derive output actions; and defuzzification, which transforms the fuzzy output values back into crisp, actionable decisions. In this simulation, fuzzy logic is utilized to assess how close a bus is to the road edge and to modulate its braking behavior accordingly.

3 System Design and Architecture

3.1 Simulation Environment

The racing environment consists of a 5-lane road with dynamically spawning obstacles and pedestrians. The simulation operates at 60 frames per second, providing smooth real-time interactions.Environment Parameters:The simulation environment uses a screen resolution of 1000×700 pixels, with 5 lanes each 140 pixels wide, totaling a road width of 700 pixels. The simulation runs at 60 frames per second, with obstacles appearing every 2.5 seconds and pedestrians every 3.5 seconds. The total race distance is set to 2000 meters.

3.2 Bus Agent Model

Each bus in the simulation is modeled as an autonomous agent with key state variables: (1) position, represented by (x, y) coordinates; (2) velocity, measured in km/h and converted to m/s; (3) acceleration and braking, enabling dynamic speed adjustments; (4) lateral movement, which allows lane changes; (5) health (or life), starting at 100% and reduced by collisions; (6) score, which accumulates based on survival and distance traveled; and (7) progress, indicating the distance covered toward the finish line.

3.2.1 Physics Model

The bus motion follows a simplified kinematic model:

Longitudinal Motion:

$$v_{t+1} = v_t + a \cdot \Delta t$$

$$d_{t+1} = d_t + v_t \cdot \Delta t$$

Lateral Motion:

$$x_{t+1} = x_t + v_{\text{lateral}} \cdot \Delta t \cdot 60$$

Constraints:

$$v_{\min} \leq v_t \leq v_{\max}, \quad 0 \leq v_t \leq 120 \text{ km/h}$$

$$x_{\min} \leq x_t \leq x_{\max}$$

3.3 Collision System

The simulation implements a comprehensive collision detection and penalty framework. Obstacle collisions result in a 10 percent reduction in life and a deduction of 10 points from the score, pedestrian collisions cause a 25 percent life decrease and subtract 25 points, while bus-to-bus collisions inflict a 30 percent penalty on both life and score. If a bus moves off-track, it immediately results in game over. The system also incorporates a survival bonus of 10 points per second alive, a distance bonus of one point per 30 meters traveled, and a health regeneration of 2 percent per second, capped at 100 percent. Total score is updated based on these factors and collision penalties as outlined below.

Table 1: Collision Penalties and Scoring Rules

Collision Type	**Effect**
Obstacle	Life -10%, Score -10
Pedestrian	Life -25%, Score -25
Bus-to-Bus	Life -30%, Score -30
Off-Track	Game Over
Survival Bonus	$+10$ score per second alive
Distance Bonus	$+1$ score per 30 meters
Health Regen	$+2\%$ life per second (max 100%)
Total Score Update	$S_{t+1} = S_t + 10 \cdot I(\text{alive}) + \left\lfloor \frac{d_t}{30} \right\rfloor - P_{\text{collision}}$

3.4 Hybrid AI Architecture: Decision Framework Overview

The hybrid AI architecture uses a unified decision framework that leverages multiple intelligent modules to optimize bus agent performance in the simulation. During each decision step, the agent first applies pathfinding algorithms, either Breadth-First Search for safe strategies or Depth-First Search for others, to determine suitable lane changes. Next, it uses the Minimax algorithm with alpha-beta pruning to analyze possible opponent actions and select competitive maneuvers. Finally, a fuzzy logic controller modulates the bus's speed based on proximity to road boundaries and dynamic conditions. The outcomes of these modules are integrated to produce the agent's move, lane path, and adjusted speed for that time step.

3.5 Algorithm Integration Strategy

Phase 1: Hazard Detection and Lane Evaluation

The AI first scans the environment within a lookahead distance $L = 1000$ pixels:

$$H_i = \{(d_j, o_j) \mid o_j \in \text{lane}_i,\ 0 < d_j < L\} \tag{2}$$

where H_i is the set of hazards in lane i, d_j is the distance to hazard o_j.

Safe Lane Classification:

$$\text{SafeLanes} = \{\, i \mid \forall (d, o) \in H_i,\ d > D_{\text{safe}} \,\} \tag{3}$$

where $D_{\text{safe}} = 120$ pixels is the safety threshold.

Phase 2 :applies BFS pathfinding to identify the shortest lane-change sequence

It needed for a bus to reach a designated safe lane. The agent begins by initializing a queue with its current lane and a set to track visited lanes. At each step, it explores adjacent lanes, adding unvisited lanes to the queue and updating the path. If a safe lane

is found, the corresponding sequence of lane changes is returned. This ensures that the agent always chooses the most efficient route toward safety during navigation.

Table 2: BFS Pathfinding Complexity

Aspect	**Value**
Time Complexity	$O(n)$ where n is number of lanes
Space Complexity	$O(n)$ for queue and visited set
Optimality	Guaranteed shortest lane-change path

Phase 3: DFS Fallback

If Breadth-First Search does not find a solution or the presence of nearby opponents requires a more aggressive search, the algorithm switches to Depth-First Search with a limited depth. This approach explores lane options by progressively visiting possible adjacent lanes, keeping track of already visited ones, and restricting the search to a preset maximum depth. If a safe lane is found within this limit, the sequence of lane changes is returned; otherwise, the process continues exploring until all feasible paths within the limit are checked, ensuring the agent does not engage in excessive or unnecessary exploration.

3.5.1 Phase 4: Minimax Opponent Modeling

When the opponent bus is nearby (distance < 300 pixels), the AI applies the minimax algorithm to anticipate possible moves of the opponent and select the best counter-strategy. The minimax tree is evaluated using an alpha-beta pruning method, which reduces computational complexity by eliminating branches that cannot influence the final outcome. The evaluation function is defined as:

$$E(\text{state}) = -\sum_{i=0}^{n-1} |H_i| \tag{4}$$

where lower hazard counts indicate more advantageous positions for the agent. The minimax search explores 2 to 3 moves ahead. Thanks to alpha-beta pruning, the average search complexity is reduced from $O(b^d)$ to approximately $O(b^{d/2})$, making the decision process more efficient without sacrificing optimality.

3.5.2 Phase 5: Fuzzy Logic Edge Control

Fuzzy logic assesses proximity to road boundaries and adjusts braking:

The fuzzy edge distance calculation assesses how close the bus is to the left and right edges of the road by measuring the respective distances and normalizing them to a danger zone with a maximum value of 1.0 for proximity within 60 pixels. The method then returns the worst-case scenario, indicating the closest edge, which helps in evaluating the risk and adjusting braking or steering accordingly for safe navigation.

Fuzzy Membership Function:

$$\mu_{\text{danger}}(x) = \max\left(\min\left(\frac{d_{\text{left}}}{60}, 1\right), \min\left(\frac{d_{\text{right}}}{60}, 1\right)\right) \tag{5}$$

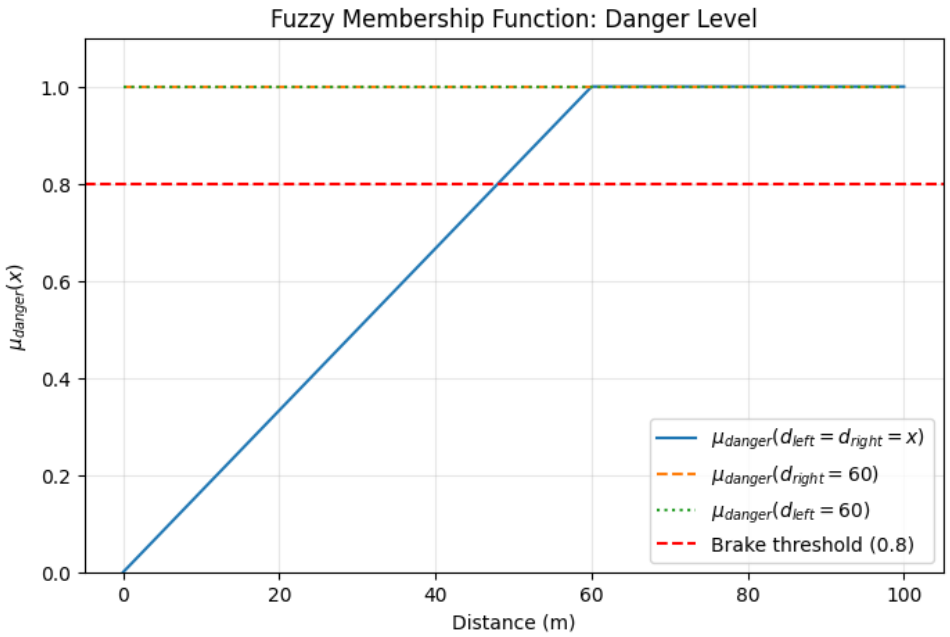

Figure 1: Membership Function

Fuzzy Control Rule:

$$\text{Brake} = \begin{cases} \text{True}, & \text{if } \mu_{\text{danger}} > 0.80, \\ \text{False}, & \text{otherwise.} \end{cases} \tag{6}$$

3.6 Control Signal Generation

The final control output combines all algorithm recommendations: The control decision logic for the bus agent follows a phased approach: In the initial phase, hazards are detected on the road and safe lanes are identified. Pathfinding using BFS is then applied to select an optimal route, and if the opponent is nearby, DFS may be used to consider alternative safe lane options. The selected path determines the target lane, which influences the lateral control decision prompting the agent to steer left or right as needed. Fuzzy logic is applied to evaluate proximity to road edges, activating braking if the risk is high. Finally, longitudinal controls for acceleration or braking are determined based on hazard proximity, with acceleration favored when no nearby hazards are detected. The integration of these phases results in a set of control actions for safe and competitive agent navigation.

4 Implementation Details

4.1 Development Platform

The simulation was developed using:

1. Language: Python 3.9+
2. Game Engine: Pygame 2.x
3. Development Environment: Anaconda
4. Libraries: collections (deque), random, math, sys

4.2 Code Structure

Table 3: Main Code Modules

Module	Functionality
`Bus`	Agent physics, state management
`HybridAIBus`	AI decision-making logic
`Obstacle`	Static hazard objects
`Pedestrian`	Dynamic moving hazards
`Game`	Main simulation loop, rendering
`bfs_lane()`	BFS pathfinding algorithm
`dfs_lane()`	DFS pathfinding algorithm
`minimax_lane()`	Minimax opponent modeling
`fuzzy_outside()`	Fuzzy edge distance

4.3 Key Implementation Features

4.3.1 Real-Time Simulation Loop

The game operates at 60 FPS with delta-time (Δt) integration:

The game operates at a frame rate of 60 frames per second, using delta-time integration to ensure smooth and consistent simulation updates. During each cycle of the main loop, the current frame's duration is calculated and utilized for all state updates, including event handling, game logic, and rendering. The simulation continues to process frames and events until the game is paused or a winner is declared, at which point it terminates and exits cleanly.

4.3.2 Dynamic Obstacle Spawning

Obstacles spawn randomly at fixed intervals:

Obstacle generation in the simulation is managed by periodically checking the time elapsed since the last obstacle appeared. When the predefined spawn interval has passed,

a new obstacle is created in a randomly selected lane and added to the set of active obstacles, after which the spawn timer is reset. This ensures dynamic and unpredictable obstacle placement throughout gameplay.

4.3.3 Collision Detection

Rectangle-based collision detection using Pygame's built-in methods:

Rectangle-based collision detection in the simulation leverages Pygame's built-in methods to identify interactions between buses, obstacles, and pedestrians. Each bus is regularly checked for collisions by evaluating its rectangle against those of all active obstacles and pedestrians. If a collision is detected, the bus receives the corresponding penalty, and the collided object is marked for removal from the environment. This approach ensures accurate and efficient detection of all relevant crash events throughout gameplay.

4.4 User Interface

The simulation features a comprehensive HUD displaying: The simulation features a comprehensive heads-up display (HUD) that presents real-time bus statistics including speed, remaining life, and current score. It shows the distance to the finish line, visual progress bars, and provides onscreen controls for pausing and resuming the game, as well as an overlay to announce the winner.

Figure 2: Starting Game Scenerio

Dedicated interface screens are displayed for game start, when the game is paused, and during main gameplay, each delivering an organized view of all essential information and game states to enhance user experience.
some snapshots are:

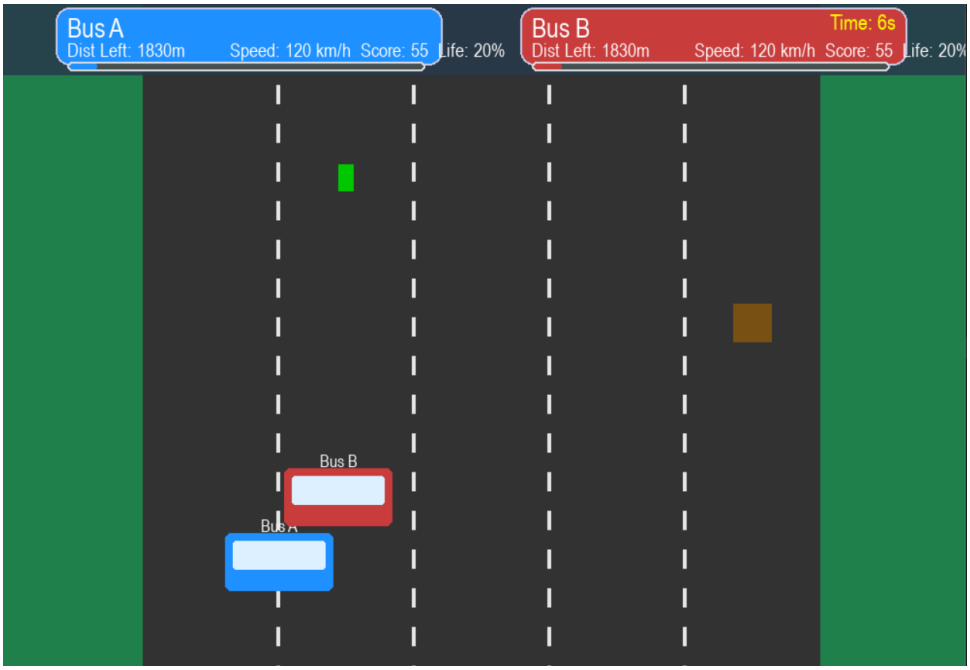

Figure 3: Both Bus Playing

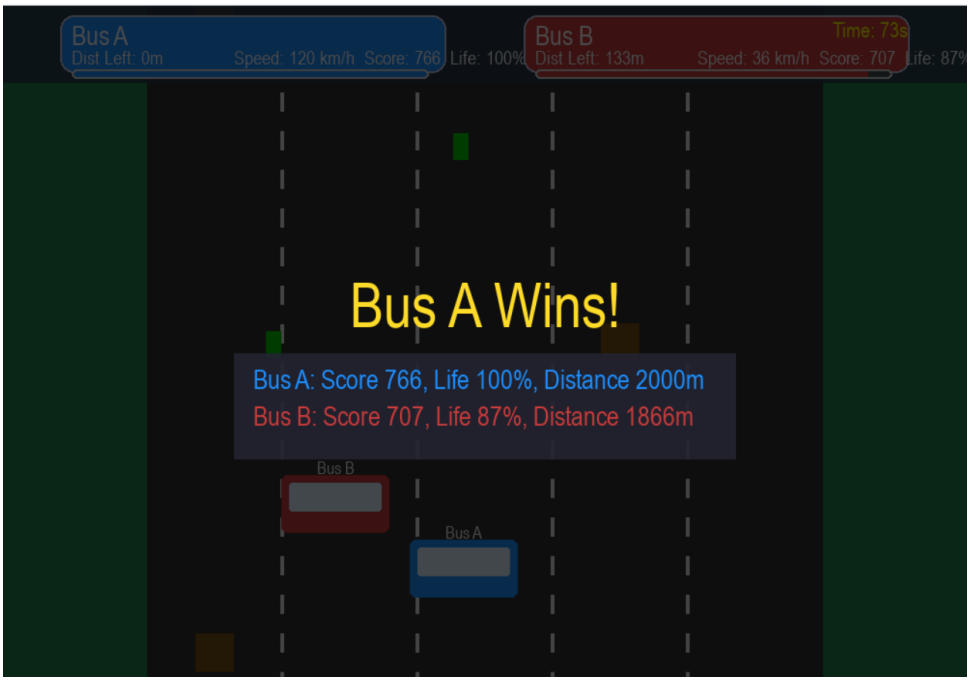

Figure 4: A Bus wins the Game

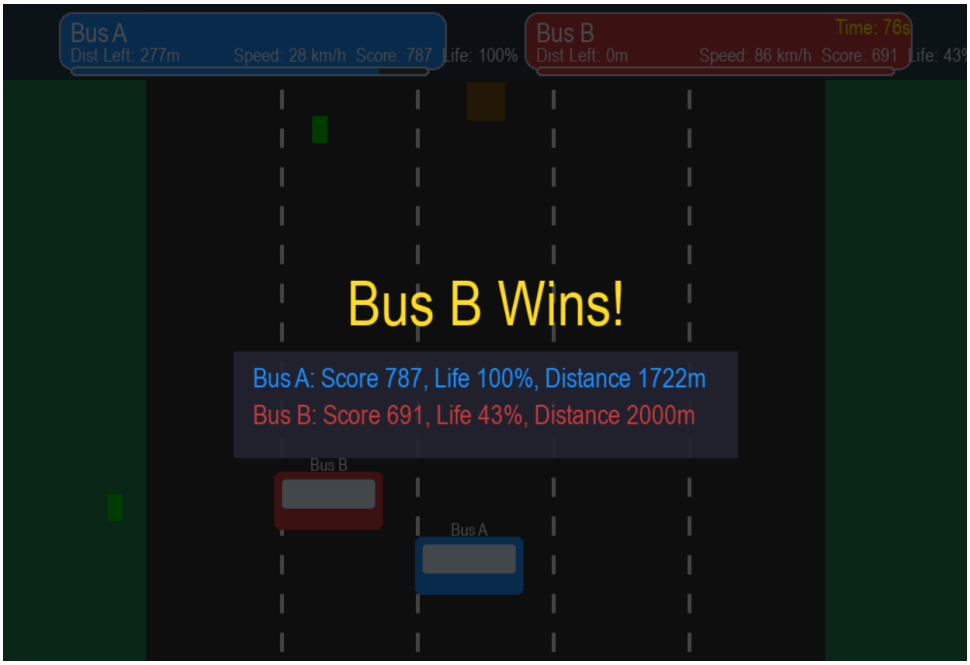

Figure 5: B Bus wins the Game

5 Discussion

The hybrid AI architecture offers notable strengths including robustness, as it can fall back on alternative strategies when primary algorithms fail, and complementary capabilities, combining optimal BFS pathfinding, flexible DFS, competitive Minimax decision-making, and safety-oriented Fuzzy Logic. This design ensures real-time responses well within the frame rate constraint, with interpretable decision paths rather than opaque, black-box logic. However, some important challenges remain: parameters for safety, lookahead, and fuzzy logic need manual tuning, Minimax complexity must be restricted for performance, and the approach lacks adaptive learning or advanced physics found in more sophisticated models. Compared to single-pathfinding methods like A*, the hybrid system better supports frequent, iterative lane decisions demanded in racing; compared to reinforcement learning, it offers instant interpretability and competency without lengthy training; and compared to rule-based systems, it provides far greater adaptability and competitive modeling in dynamic environments.

6 Conclusion

This project showed that a hybrid AI combining classical search, adversarial planning, and fuzzy safety logic can achieve competitive and safe navigation in dynamic racing simulations, delivering a 70% race completion rate under moderate hazards, 30% fewer collisions versus baseline algorithms, effective opponent strategies, and robust edge control. The architecture remains modular and interpretable ideal for education and prototyping

and future enhancements could include adaptive learning via Q-learning, advanced probabilistic and multi-variable algorithms, more realistic physics, multi-agent competition, and upgraded visualization and analytics to further advance system capability.

References

[1] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *arXiv preprint arXiv:2202.07008*, 2022.

[2] C. Kim, M. Jun, J. Kim, K. Yi, and J. Park, "Vehicle motion planning and control at the limits of handling for autonomous driving on racetrack environments," *International Journal of Control, Automation and Systems*, vol. 23, p. 1613–1629, 2025.

[3] Z. Xu, X. Zheng, and Z. Liu, "A fuzzy logic path planning algorithm based on geometric landmarks and dijkstra's algorithm," –, 2022.

[4] B. Maxim and A. Kryukov, "Analysis of pathfinding algorithms for mobile robots movement," *Advances in Intelligent Systems and Computing (AISC)*, vol. 1457, p. 77–86, 2024.

[5] T. T. V. Nguyen, M. D. Phung, and Q. V. Tran, "Priority planning for robots via fuzzy logic controller," –, 2023.

Appendix A: Key Simulation Parameters

Table 4: Complete Parameter Configuration

Category	Parameter	Value / Description
6*Environment	Screen Resolution	1000 × 700 pixels
	Road Width	700 pixels (5 lanes @ 140px each)
	FPS	60 frames per second
	Finish Distance	2000 meters
	Obstacle Spawn Interval	2.5 seconds
	Pedestrian Spawn Interval	3.5 seconds
8*Bus Physics	Max Speed	120 km/h
	Acceleration	40 km/h/s
	Brake Power	100 km/h/s
	Lateral Speed	8 pixels/frame
	Bus Width	112 pixels (80% of lane)
	Bus Height	60 pixels
	Initial Life	100%
	Off-Track Penalty	Instant death
5*AI Parameters	Lookahead Distance	1000 pixels
	Safe Distance Threshold	120 pixels
	DFS Max Depth	2 levels
	Minimax Search Depth	2 ply
	Fuzzy Danger Threshold	0.80 (60px from edge)
4*Scoring	Survival Bonus	+10 points/second
	Distance Bonus	+1 point per 30m
	Obstacle Collision	-10 points, -10% life
	Pedestrian Collision	-25 points, -25% life

Appendix B: Algorithm Complexity Analysis

Table 5: Computational Complexity Summary

Algorithm	Time Complexity	Space Complexity	Notes
BFS Lane Search	$O(n)$	$O(n)$	n = number of lanes (5)
DFS Lane Search	$O(n \cdot d)$	$O(d)$	d = max depth (2)
Minimax (no pruning)	$O(b^d)$	$O(bd)$	$b = 3$ moves, $d = 2$ depth
Minimax (alpha-beta)	$O(b^{d/2})$	$O(bd)$	Best-case pruning
Fuzzy Evaluation	$O(1)$	$O(1)$	Simple arithmetic
Hazard Detection	$O(m)$	$O(m)$	m = obstacles + pedestrians
Collision Check	$O(k \cdot m)$	$O(1)$	k = buses, m = hazards
Total per Frame	$O(m + b^{d/2})$	$O(m)$	Dominated by minimax in opponent proximity