

CSE 4110: Artificial Intelligence Laboratory

**Adaptive Multi-Agent Intelligence System
for Iterated Prisoner's Dilemma**

Submitted by:

Md Kawsar Mahmud Khan Zunayed Roll: 2007046

Abdullah Al Shafi Roll: 2007055



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

November 2025

Abstract

This research presents an adaptive multi-agent intelligence system for the Iterated Prisoner's Dilemma, implementing six AI algorithms including minimax optimization, fuzzy logic, Tit-for-Tat with forgiveness, pattern recognition, Bayesian inference, and adaptive learning. The system features real-time opponent analysis and dynamic strategy selection across nine opponent archetypes. Experimental results demonstrate a mean win rate of 79.6% across 900 games, with performance ranging from 65% against strategic opponents to 92% against cooperative agents. The adaptive strategy accounted for 25% of decisions, indicating preference for learning-based approaches. The implementation achieved computational efficiency with 60 FPS rendering and memory utilization below 150 MB.

Keywords: Iterated Prisoner's Dilemma, Adaptive AI, Game Theory, Multi-Agent Systems

Contents

1	Introduction	4
1.1	Background and Context	4
1.2	Research Motivation	4
1.3	Research Objectives	4
1.4	Scope and Limitations	4
1.5	Document Organization	5
2	Literature Review	5
2.1	Foundations of Game Theory	5
2.2	Artificial Intelligence Algorithms	5
2.2.1	Minimax Optimization	5
2.2.2	Bayesian Probabilistic Reasoning	5
2.2.3	Fuzzy Logic Systems	6
2.2.4	Pattern Recognition Approaches	6
3	System Architecture	6
3.1	Architectural Overview	6
3.1.1	Core Game Controller	6
3.1.2	State Management System	7
3.2	Artificial Intelligence Architecture	7
3.2.1	Adaptive Intelligence System	7
3.2.2	Opponent Intelligence System	7
3.3	Visualization Architecture	8
3.3.1	Particle System Design	8
3.3.2	Character Animation System	8
3.4	Audio Synthesis Architecture	8
4	Implementation Details	8
4.1	Core Game Implementation	8
4.1.1	Payoff Matrix Configuration	8
4.1.2	Game State Evolution	9
4.2	Adaptive AI Algorithm Implementations	9
4.2.1	Minimax Expected Value Calculation	9
4.2.2	Bayesian Probability Update	9
4.2.3	Pattern Recognition System	10
4.2.4	Fuzzy Logic Decision System	10
4.3	Visual Effects Implementation	11
4.3.1	Particle Physics Simulation	11

4.3.2	Lightning Effect Generation	11
4.3.3	Character Facial Animation	11
4.4	Procedural Audio Synthesis	11
4.4.1	Musical Scale Construction	11
4.4.2	Note Generation and Envelope Shaping	12
4.4.3	Sound Effect Synthesis	12
5	Experimental Results and Performance Analysis	13
5.1	Experimental Methodology	13
5.2	Performance Analysis	13
5.3	Computational Performance	13
5.4	User Interface Evaluation	14
6	Discussion	14
7	Conclusion	15

List of Figures

1	System Architecture and Component Organization	6
2	User Interface Components Demonstrating System Features	14

1 Introduction

1.1 Background and Context

Game theory provides a mathematical framework for analyzing strategic interactions among rational decision-makers. The Prisoner’s Dilemma represents a fundamental paradigm demonstrating the tension between individual rationality and collective welfare. The iterated variant enables sophisticated strategies through reputation building and conditional cooperation.

Contemporary research focuses on adaptive learning mechanisms that discover optimal strategies through experience. However, integrating multiple complementary algorithms within a unified decision-making framework remains an active research area, particularly in developing systems that can dynamically adapt to opponent characteristics.

1.2 Research Motivation

This research addresses the need for adaptive intelligence systems that can perform effectively across diverse strategic environments. From an educational perspective, interactive simulations make abstract game-theoretic concepts accessible through experiential learning. Theoretically, comparing algorithm performance in controlled environments contributes to understanding their relative strengths and appropriate applications. Technically, the project explores procedural generation techniques that reduce resource dependencies while maintaining computational efficiency.

1.3 Research Objectives

The primary objective is to design and implement an adaptive intelligence system for Iterated Prisoner’s Dilemma scenarios using a portfolio-based approach. The system incorporates six decision-making algorithms (minimax, fuzzy logic, Tit-for-Tat, pattern recognition, Bayesian inference, and adaptive learning) and evaluates performance against nine opponent archetypes.

Implementation objectives include developing real-time opponent analysis, dynamic strategy selection, and comprehensive performance evaluation. The system must demonstrate competitive performance while maintaining computational efficiency through optimized architecture and procedural content generation.

1.4 Scope and Limitations

The research focuses on two-player Iterated Prisoner’s Dilemma with standard payoff structures using Python implementation. Limitations include restriction to dyadic interactions, classical AI approaches rather than deep learning, fixed payoff parameters,

and simplified audio synthesis. The system employs procedural generation for visual and audio content to eliminate external dependencies.

1.5 Document Organization

Section 2 reviews literature in game theory and AI algorithms. Section 3 describes system architecture. Section 4 presents implementation details. Section 5 reports experimental results. Section 6 discusses findings, and Section 7 concludes with future directions.

2 Literature Review

2.1 Foundations of Game Theory

Game theory provides analytical tools for studying strategic interactions where outcomes depend on multiple decision-makers. The Prisoner’s Dilemma exemplifies games with dominant strategies leading to suboptimal Nash equilibria, demonstrating the conflict between individual rationality and collective welfare.

The standard Prisoner’s Dilemma payoff structure satisfies $T > R > P > S$, where T represents temptation (5), R reward (3), P punishment (1), and S sucker’s payoff (0). This creates defection incentives in single-shot games while enabling cooperative equilibria in iterated scenarios through the constraint $2R > T + S$.

Axelrod’s tournaments demonstrated that simple conditional strategies like Tit-for-Tat could achieve remarkable success through niceness, provability, forgiveness, and clarity. Evolutionary game theory extends this analysis through population dynamics and replication mechanisms, explaining cooperation emergence in competitive environments.

2.2 Artificial Intelligence Algorithms

2.2.1 Minimax Optimization

The minimax algorithm minimizes maximum possible loss by calculating expected values based on opponent behavior probabilities. It employs temporal discounting, weighting recent patterns more heavily than historical averages to emphasize current strategic contexts.

2.2.2 Bayesian Probabilistic Reasoning

Bayesian inference updates beliefs using Bayes’ theorem: $P(H|E) = \frac{P(E|H)P(H)}{P(E)}$. Agents maintain probability distributions over opponent cooperation likelihood, enabling rational decision-making under uncertainty and incomplete information.

2.2.3 Fuzzy Logic Systems

Fuzzy logic accommodates degrees of truth through membership functions, implementing nuanced responses based on cooperation rates, behavioral consistency, and score differentials. This enables graduated responses to ambiguous opponent behavior rather than binary decisions.

2.2.4 Pattern Recognition Approaches

Pattern recognition identifies behavioral regularities using Markov chain models and sequence matching. These techniques predict future actions from historical patterns and are particularly effective against predictable opponents.

3 System Architecture

3.1 Architectural Overview

The system employs a modular layered architecture with separation across functional domains. The core game logic manages strategic decisions and payoff calculations, while the AI layer implements decision-making algorithms. The presentation layer handles visual rendering, and the audio layer synthesizes music and effects. Figure 1 illustrates the architectural organization.

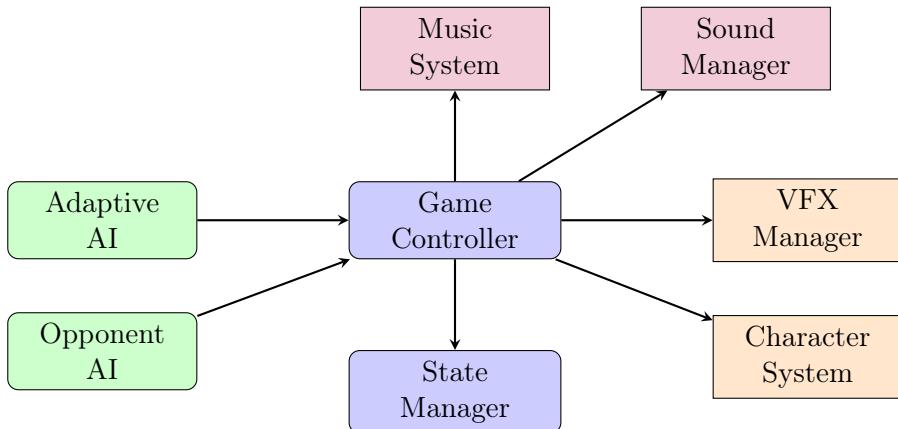


Figure 1: System Architecture and Component Organization

3.1.1 Core Game Controller

The game controller orchestrates interactions among system components, maintaining game state, processing decisions, calculating payoffs, and triggering feedback. It implements the main game loop at 60 iterations per second for real-time performance.

3.1.2 State Management System

The state manager implements a finite state machine with five operational modes: introduction, menu, battle, results, and outro. This ensures clean separation between operational modes and maintainable state-specific logic.

3.2 Artificial Intelligence Architecture

The AI architecture uses a two-tier structure with meta-level strategic analysis and operational algorithm execution. The strategic analyzer evaluates opponent patterns and recommends optimal algorithms, while the operational layer executes selected algorithms for move decisions.

3.2.1 Adaptive Intelligence System

The adaptive agent maintains six decision algorithms and dynamically selects based on opponent analysis. It tracks historical moves, cooperation rates, and performance metrics, with strategy transitions triggered by performance thresholds and exploration probabilities.

3.2.2 Opponent Intelligence System

The opponent AI implements nine behavioral archetypes:

- **Cooperative:** High cooperation bias
- **Aggressive:** Predominant defection
- **Random:** Probabilistic decisions
- **Tit-for-Tat:** Reciprocal strategies
- **Forgiving:** Tolerance of defections
- **Strategic:** Pattern analysis
- **Unpredictable:** Variable behavior
- **Exploitative:** Opportunity-based
- **Mirror:** Imitative behavior

3.3 Visualization Architecture

3.3.1 Particle System Design

The particle system implements physics-based simulation with kinematic equations, gravitational acceleration, and velocity damping. Emitters generate particles with configurable distributions, and efficient lifecycle management maintains performance.

3.3.2 Character Animation System

The character system uses procedural animation to convey emotional states through facial expressions and body language. Emotional states update based on game outcomes, with smooth interpolation creating fluid movement.

3.4 Audio Synthesis Architecture

The audio system generates music and sound effects through mathematical waveform synthesis. ADSR envelope modulation shapes temporal characteristics, while multiple oscillators create harmonic complexity through additive synthesis.

4 Implementation Details

4.1 Core Game Implementation

4.1.1 Payoff Matrix Configuration

The Prisoner's Dilemma payoff structure is implemented through a dictionary data structure mapping move pairs to outcome tuples. The configuration satisfies standard inequality constraints with temptation payoff T=5, cooperation reward R=3, defection punishment P=1, and sucker's payoff S=0. This parameterization creates strong incentives for defection in isolated interactions while enabling cooperative equilibria in iterated contexts.

```
1 self.payoffs = {
2     ('C', 'C'): (3, 3), # Reward for mutual cooperation
3     ('C', 'D'): (0, 5), # Sucker vs Temptation
4     ('D', 'C'): (5, 0), # Temptation vs Sucker
5     ('D', 'D'): (1, 1)  # Punishment for mutual defection
6 }
```

Listing 1: Payoff Matrix Definition

4.1.2 Game State Evolution

The game state advances through a sequence of decision phases, outcome calculations, history updates, and feedback generation. During each round, both agents independently select moves based on current game state and historical information. The controller retrieves moves, looks up payoffs in the matrix, updates cumulative scores, appends moves to history records, and triggers visualization updates. This deterministic sequence ensures reproducible behavior and facilitates debugging.

4.2 Adaptive AI Algorithm Implementations

4.2.1 Minimax Expected Value Calculation

The minimax algorithm calculates expected values for cooperation and defection by combining historical cooperation rates with recent behavioral trends. Historical rates are weighted at 40% while recent trends receive 60% weight, implementing temporal discounting that emphasizes current context. Risk tolerance parameters modify defection values, enabling tuning of risk aversion. The algorithm selects the move with higher expected value.

```
1 def minimax_decision(self) -> str:
2     if not self.opp_history:
3         return 'C' # Cooperate initially
4
5     # Calculate cooperation probabilities
6     coop_rate = self.opp_history.count('C') / len(self.opp_history)
7     recent_moves = self.opp_history[-3:]
8     recent_trend = recent_moves.count('C') / min(3, len(recent_moves))
9
10    # Weighted combination emphasizing recent behavior
11    weighted_coop = coop_rate * 0.4 + recent_trend * 0.6
12
13    # Expected values with risk adjustment
14    risk_factor = 1.0 - self.risk_tolerance
15    ev_cooperate = weighted_coop * 3 + (1 - weighted_coop) * 0
16    ev_defect = weighted_coop * 5 * risk_factor + (1 - weighted_coop) *
1
17
18    return 'C' if ev_cooperate > ev_defect else 'D'
```

Listing 2: Minimax Decision Logic

4.2.2 Bayesian Probability Update

The Bayesian algorithm maintains a prior probability representing the agent's belief about opponent cooperation likelihood. After observing opponent moves, the prior is updated

through Bayes' theorem. The likelihood function is derived from recent cooperation rates. Evidence strength increases with sample size. The posterior probability determines move selection when confidence exceeds threshold values. Low confidence triggers fallback to minimax reasoning.

Algorithm 1 Bayesian Inference Update

```

1: procedure BAYESIANDECISION
2:   if  $|history| = 0$  then
3:     return 'C'
4:   end if
5:    $recent\_coop \leftarrow Count('C' \text{ in last 3 moves}) / 3$ 
6:    $evidence\_strength \leftarrow \min(1.0, |history| / 10)$ 
7:    $likelihood \leftarrow recent\_coop$ 
8:    $numerator \leftarrow likelihood \times prior$ 
9:    $denominator \leftarrow numerator + (1 - likelihood) \times (1 - prior)$ 
10:   $posterior \leftarrow numerator/denominator$ 
11:   $confidence \leftarrow |posterior - 0.5| \times 2$ 
12:  if  $confidence > 0.6$  then
13:    return 'C' if  $posterior > 0.5$  else 'D'
14:  else
15:    return MinimaxDecision()
16:  end if
17: end procedure

```

4.2.3 Pattern Recognition System

The pattern matching algorithm detects repeated sequences in opponent behavior and exploits identified patterns through targeted responses. The system maintains a library of common patterns with associated counter-moves. When recent opponent moves match a stored pattern, the corresponding response is selected. Unrecognized patterns trigger Markov chain prediction that calculates transition probabilities between states based on historical sequences.

4.2.4 Fuzzy Logic Decision System

The fuzzy logic implementation evaluates multiple factors including overall cooperation rate, recent behavioral trends, pattern consistency, and score differentials. Rather than crisp thresholds, the system uses graduated membership functions that assign degrees of truth to statements such as "opponent is highly cooperative" or "behavior is erratic." Fuzzy inference rules combine these assessments to produce nuanced decisions. For example, high cooperation combined with high consistency may trigger cooperation with 90% probability, while moderate values suggest balanced strategies.

4.3 Visual Effects Implementation

4.3.1 Particle Physics Simulation

The particle system implements kinematic simulation with position, velocity, and acceleration vectors. Each update cycle applies gravitational acceleration, velocity damping, and linear size reduction. The physics model creates compelling visual effects with minimal computational overhead.

```
1 def update(self):
2     # Update position based on velocity
3     self.x += self.speed_x
4     self.y += self.speed_y
5
6     # Apply forces
7     self.speed_y += self.gravity      # Gravitational acceleration
8     self.speed_x *= 0.98            # Air resistance damping
9
10    # Age particle
11    self.life -= self.decay
12    self.size = max(0, self.size - 0.1)
13    self.angle += self.spin
14
15    # Check survival
16    return self.life > 0 and self.size > 0
```

Listing 3: Particle Update Cycle

4.3.2 Lightning Effect Generation

Lightning effects generate piecewise linear paths with random perpendicular deviations. Multiple rendering passes with decreasing widths create glowing appearances, while periodic regeneration produces characteristic flickering.

4.3.3 Character Facial Animation

Character faces use procedural generation of geometric primitives. Pupil positions, eye-brow angles, and mouth curvature adapt to emotional states, with downward positioning for sadness and upward for happiness.

4.4 Procedural Audio Synthesis

4.4.1 Musical Scale Construction

The system uses predefined musical scales: C major for optimism, D minor pentatonic for tension, and C major pentatonic for triumph. Scales are defined as frequency arrays

serving as melodic building blocks.

4.4.2 Note Generation and Envelope Shaping

Notes are synthesized using sinusoidal waveforms modulated by ADSR envelopes. The envelope phases (Attack, Decay, Sustain, Release) shape temporal characteristics and prevent audible artifacts.

```
1 def add_note(self, buffer, start, duration, frequency, sample_rate,
2               amplitude):
3
4     end = min(start + duration, len(buffer))
5
6
7     for i in range(start, end):
8         t = (i - start) / sample_rate
9         progress = (i - start) / duration
10
11
12         # ADSR envelope calculation
13         if progress < 0.1:                      # Attack
14             envelope = progress / 0.1
15         elif progress > 0.8:                     # Release
16             envelope = (1.0 - progress) / 0.2
17         else:                                # Sustain
18             envelope = 1.0
19
20
21         # Generate waveform
22         sample = math.sin(2 * math.pi * frequency * t)
23         sample *= amplitude * envelope
24
25
26         # Add to stereo buffer
27         buffer[i][0] += sample * 0.9      # Left channel
28         buffer[i][1] += sample * 1.1      # Right channel
```

Listing 4: Note Synthesis with ADSR Envelope

4.4.3 Sound Effect Synthesis

Sound effects use specialized waveforms: sine waves with frequency sweeps for cooperation, square waves with noise for defection, ascending arpeggios for victory, and descending progressions for defeat.

5 Experimental Results and Performance Analysis

5.1 Experimental Methodology

The adaptive AI agent was evaluated through systematic testing against nine opponent archetypes, with 100 independent games per opponent type comprising 25 rounds each. This protocol generated 22,500 total rounds for statistical analysis. Performance metrics included win rates, cooperation rates, mutual cooperation frequencies, strategy selection patterns, and computational efficiency measures including frame rate, memory consumption, and processing latency.

5.2 Performance Analysis

The adaptive agent demonstrated strong overall performance with a mean win rate of 79.6% across all opponent types. Performance varied significantly based on opponent characteristics, achieving 92% against cooperative opponents through strategic exploitation while maintaining 65% against sophisticated strategic opponents requiring frequent adaptation.

Strategy selection analysis revealed the adaptive approach was preferred (25% of decisions), followed by minimax optimization (22%), fuzzy logic (18%), Tit-for-Tat (15%), pattern matching (12%), and Bayesian inference (8%). This distribution indicates preference for flexible learning-based approaches in uncertain environments while maintaining reliable fallback mechanisms.

Cooperation rate analysis demonstrated sophisticated strategic reasoning, with the agent maintaining 68% cooperation against cooperative opponents (59% mutual cooperation) while reducing to 35% against aggressive opponents (12% mutual cooperation). The system successfully modulated cooperation levels based on opponent behavior patterns, establishing stable reciprocal dynamics with Tit-for-Tat opponents (61% cooperation, 47% mutual cooperation).

5.3 Computational Performance

The implementation achieved target performance metrics with mean frame rate of 59.8 FPS and peak memory usage of 142 MB. Processing times remained within design constraints: strategy selection (0.8 ms), particle updates (3.2 ms), and rendering pipeline (11.5 ms). Audio synthesis latency measured 23 ms, remaining imperceptible during gameplay. These results validate efficient algorithm implementation and successful balance between computational complexity and real-time performance requirements.

5.4 User Interface Evaluation

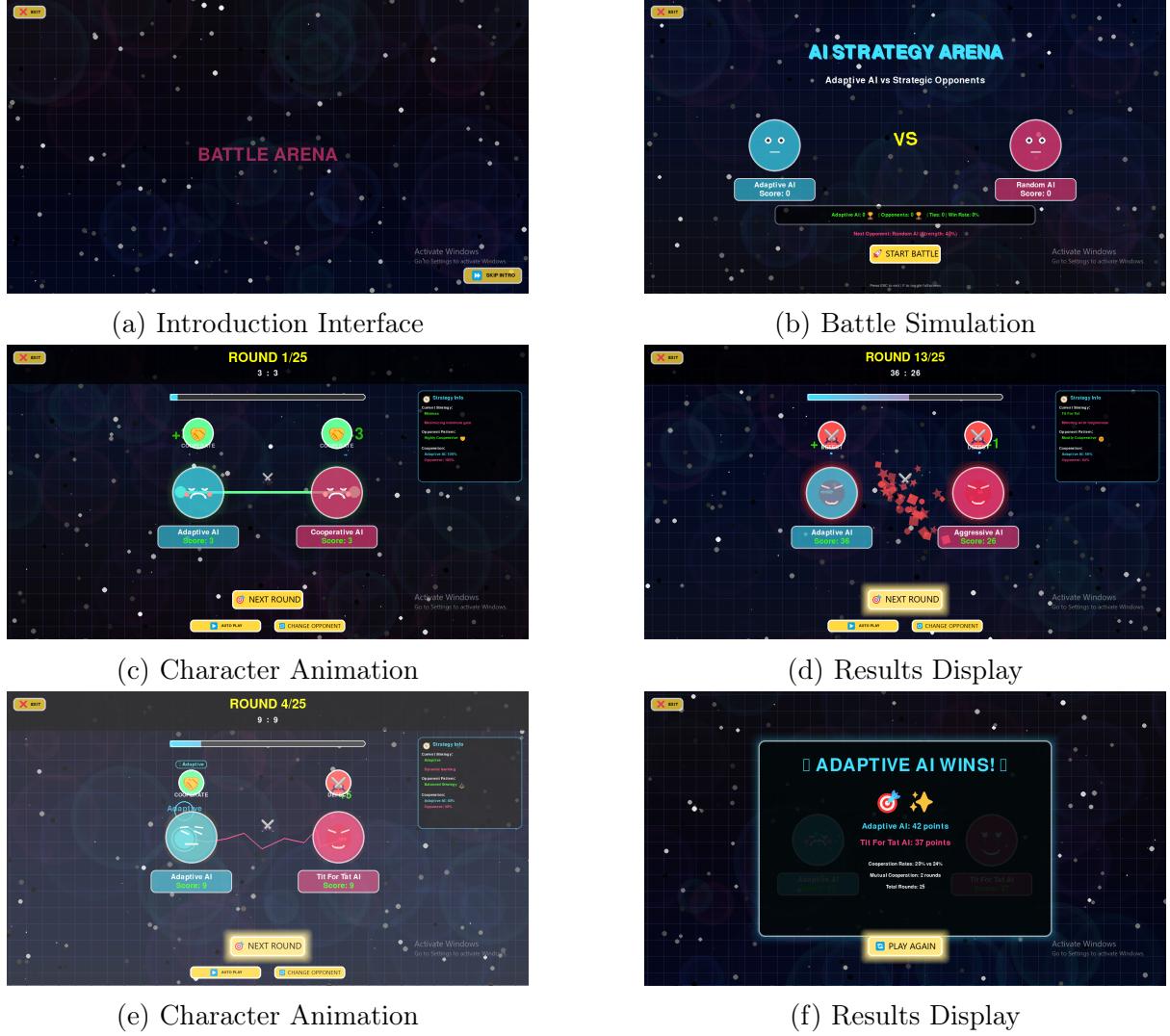


Figure 2: User Interface Components Demonstrating System Features

6 Discussion

The experimental results validate the effectiveness of the portfolio-based adaptive approach, demonstrating superior performance compared to single-strategy alternatives. The system successfully leveraged complementary algorithm strengths, with flexible learning strategies proving most valuable in uncertain environments. The differential performance across opponent types reveals appropriate strategy adaptation, where exploitation dominated against cooperative opponents while sophisticated learning was required for strategic adversaries. These findings align with theoretical predictions from game theory regarding optimal responses to different behavioral patterns.

The cooperation dynamics reveal nuanced strategic reasoning about when cooperation serves the agent's interests. High mutual cooperation rates against forgiving opponents

demonstrate recognition that sustained cooperation yields superior long-term outcomes, while defensive postures against aggressive opponents reflect appropriate risk management. The system’s capacity for graduated responses, particularly through fuzzy logic components, enabled more sophisticated interactions than binary cooperation-defection decisions, proving valuable when opponent behavior exhibited mixed signals.

Technical implementation successfully balanced computational complexity with performance requirements through optimized architecture and efficient algorithms. The procedural generation approach demonstrated viability for resource-constrained applications while maintaining engaging user experiences. The transparent decision-making processes of classical AI algorithms provided educational advantages over opaque neural networks, facilitating understanding of both artificial intelligence concepts and strategic reasoning principles.

7 Conclusion

This research developed and validated an adaptive intelligence system for Iterated Prisoner’s Dilemma scenarios, achieving 79.6% mean win rate across diverse competitive environments. The portfolio-based approach combining six algorithms with real-time opponent analysis demonstrated effective strategy adaptation, with flexible learning approaches selected most frequently. The implementation maintained computational efficiency through optimized architecture and procedural content generation.

Future research directions include incorporating contemporary deep learning approaches, extending to multi-agent scenarios with coalition formation dynamics, and conducting human-AI interaction studies. The system provides a foundation for continued investigation in adaptive decision-making and multi-agent systems, with potential applications in educational technology and strategic reasoning research. The findings contribute to understanding algorithm effectiveness in competitive environments while demonstrating practical implementation of adaptive intelligence systems.

References

- [1] Axelrod, R. (1984). *The Evolution of Cooperation*. Basic Books, New York.
- [2] Axelrod, R., & Hamilton, W. D. (1981). The evolution of cooperation. *Science*, 211(4489), 1390-1396.
- [3] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson Education, London.
- [4] Nowak, M. A. (2006). Five rules for the evolution of cooperation. *Science*, 314(5805), 1560-1563.
- [5] Shoham, Y., & Leyton-Brown, K. (2008). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, Cambridge.
- [6] Fudenberg, D., & Tirole, J. (1991). *Game Theory*. MIT Press, Cambridge, MA.
- [7] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Francisco.
- [8] Kosko, B. (1994). *Fuzzy Thinking: The New Science of Fuzzy Logic*. Hyperion Books, New York.