© Volume 2, Issue 9, ***September*** 2026 | JATIR
ISSN: 3139-8464 (p)

Shadow Tag Design and Implementation of A 2d Survival Arcade Game with A Delayed Mimicry Ai

[1]Dr.Lalji Prasad, [2]Karan Sharma
[1]*Supervised by, HOI, IAC, Department of Advance Computing Specialization, SAGE University, Indore*
[2]*Submitted by, Department of Advance Computing Specialization, SAGE University, Indore*

***Abstract*—Artificial Intelligence (AI) is increasingly applied within digital game development to build intelligent mechanics that boost player engagement, adaptability, and strategic thinking. Standard 2D arcade games typically rely on scripted enemy behavior and fixed movement routines, which makes gameplay predictable and limits replay value. This paper presents Shadow Tag, an AI-driven 2D survival arcade game built around a Delayed Mimicry AI, in which the antagonist replays the player's own past movement trajectory after a set time delay. Rather than relying on conventional pursuit logic such as pathfinding or direct target tracking, the proposed AI turns the player's earlier actions into an evolving obstacle, demanding ongoing strategic planning and spatial awareness.**

The game is built with Python and the Pygame framework using an object-oriented design that combines player control, a delayed shadow AI, procedurally generated obstacles, collectible objects, collision detection, animation handling, particle effects, persistent score tracking, and a state-driven interface. The delayed mimicry effect is produced by recording the player's position each frame and replaying those coordinates after a set delay, yielding an opponent whose behavior continually shifts in response to what the player has done. Supporting systems such as procedural level generation, real-time collision handling, high-score saving, and consistent 60 FPS execution round out an immersive, responsive play experience.

The working implementation shows that the proposed framework successfully blends intelligent AI behavior with real-time mechanics while keeping performance stable and gameplay interactive. The delayed mimicry model raises gameplay difficulty without demanding computationally heavy machine learning, giving an efficient and scalable AI approach for arcade-style titles. The architecture leaves room for future additions such as adaptive difficulty, reinforcement-learning-driven enemy behavior, multiplayer support, procedural level design, and more advanced AI decision-making. Shadow Tag thus offers a fresh way to bring behavior-driven artificial intelligence into modern 2D survival games

ISSN: 3139-8464 (p)

while showing how AI concepts can be put to practical use in interactive entertainment and game development.

***Index Terms*—Artificial Intelligence, Delayed Mimicry AI, 2D Survival Arcade Game, Shadow Tag, Python, Pygame, Game Development, Player Behavior Modelling, Real-Time Game Systems, Collision Detection, Procedural Generation, Interactive Entertainment.**

I. INTRODUCTION

1.1 Background and Motivation

Traditional 2D survival and arcade titles typically give enemies fixed paths, random movement routines, or basic pathfinding that simply chases the player's current location [11, 13]. These techniques are cheap to compute, but they tend to grow predictable with repeated play, which hurts engagement and long-term replay value.

Advances in Artificial Intelligence have made it possible to build game agents that respond dynamically to how a player behaves [2, 10]. AI-based gameplay mechanics deepen immersion by giving players opponents that change along with their choices. Yet many advanced AI approaches — reinforcement learning, deep neural networks, evolutionary algorithms — need heavy computation and elaborate training pipelines [3, 4], which makes them hard to fit into lightweight arcade games.

Shadow Tag was motivated by the goal of raising gameplay difficulty through an original AI mechanism that avoids expensive machine learning models. Rather than learning from training data, the Delayed Mimicry AI keeps a running record of the player's movement and plays it back after a fixed delay. As a result, every move the player makes shapes what happens later in the game, pushing players to think ahead about the consequences of their own actions.

The project also sets out to show that inventive AI design can meaningfully improve player experience even with fairly simple algorithms and data structures. By bringing together delayed movement replay, procedurally placed obstacles, collision detection, animation, and persistent scoring, Shadow Tag serves as a working example of how AI ideas can be woven into modern game development.

1.2 Problem Statement

Most 2D arcade survival games depend on familiar enemy behaviors — direct chasing, random wandering, or fixed movement routes [11]. These are simple to build, but they tend to yield repetitive gameplay with little strategic depth. Once players learn how the enemy behaves, the challenge fades, which lowers engagement and replay ability over time.

Building adaptive AI that produces unpredictable, intelligent gameplay usually calls for advanced machine learning methods requiring sizeable datasets, long training runs, and considerable computing power [3, 4]. These demands are frequently unrealistic for small, independently developed games built in Python and Pygame.

ISSN: 3139-8464 (p)

The core challenge tackled in this research is designing and building an AI opponent that is both intelligent and computationally light — one that keeps adapting to the player without relying on complex learning algorithms. The project examines whether replaying delayed movement can produce a genuinely challenging environment while still keeping real-time performance, straightforward implementation, and room to grow.

1.3 Research Objectives

The primary objectives of this research are:

1. Design and build a 2D survival arcade game in Python with the Pygame framework that offers an engaging, strategically demanding gameplay experience.
2. Implement a Delayed Mimicry AI that plays back the player's own movement history after a fixed time delay, forming an adaptive AI opponent.
3. Build a real-time game architecture using object-oriented programming, event handling, animation, collision detection, procedural object placement, and game-state management.
4. Implement an efficient system for recording and replaying movement that reproduces player motion accurately while keeping the frame rate steady.
5. Assess how effectively behavior-driven AI boosts player engagement, strategic thinking, and replayability relative to conventional enemy movement schemes.
6. Lay a scalable groundwork that supports later additions such as reinforcement learning, adaptive difficulty, procedural level generation, multiplayer play, and smarter path planning.

1.4 Scope and Original Contributions

This research covers the design, build, and evaluation of an AI-driven 2D survival arcade game centered on a novel delayed-mimicry mechanic. Its scope spans game architecture, AI behavior modelling, procedural environment generation, collision detection, animation, score persistence, and interface design, all implemented in Python and Pygame. The project's central focus is lightweight AI approaches that deliver intelligent gameplay without needing machine learning training or dedicated hardware.

The original contributions of this research include:

1. A new Delayed Mimicry AI design where the enemy plays back the player's own movement history rather than depending on fixed patterns or classic pathfinding [11, 13].
2. A lightweight, behavior-driven AI framework that produces adaptive, strategic gameplay without neural networks, reinforcement learning, or costly training [3, 4].
3. A real-time system for recording, storing, and replaying player positions with precise timing while keeping gameplay smooth.
4. A unified object-oriented architecture bringing together player control, delayed AI behavior, procedural obstacle generation, collision detection, animation, particle effects, persistent scoring, and a state-driven UI [28, 29].

5. A scalable framework open to future extension with more advanced AI methods — reinforcement learning, adaptive enemy behavior, procedural level generation, multiplayer play, and richer game analytics.

II. LITERATURE REVIEW

This section surveys existing work on Artificial Intelligence (AI), game development, behavior-driven enemy systems, procedural content generation, and real-time game architectures. It weighs the strengths and shortcomings of current approaches and points to the research gap that the proposed Shadow Tag system addresses.

2.1 Artificial Intelligence in 2D Game Development

Artificial Intelligence now plays a central role in modern games, enabling intelligent non-player characters (NPCs), adaptive gameplay, and believable interactions [2]. Traditional 2D games mostly lean on rule-based AI, finite state machines (FSM), and basic pathfinding to control enemies [11, 12]. These approaches are cheap to run and simple to build, but tend to become predictable with repeated play. Recent work suggests behavior-driven AI can raise engagement by making opponents feel more dynamic and responsive [10]. Even so, many AI implementations favor chasing or attacking over building gameplay mechanics that truly stem from player actions — pointing to a need for AI models that stay lightweight while still delivering a real challenge.

2.2 Behavior Modelling and Enemy Movement Algorithms

The difficulty and replay value of arcade games hinge heavily on enemy behavior. Common approaches include A* pathfinding, Breadth-First Search (BFS), waypoint navigation, and random movement [11, 13]. More sophisticated techniques — behavior trees, utility-based AI, reinforcement learning — let enemies adapt intelligently as conditions change [4, 10]. These add realism but come with heavier computation and implementation cost. Little research has looked at enemy behavior generated directly from a player's own history of actions, which is exactly the gap the proposed Delayed Mimicry AI fills, having the opponent recreate the player's past movement instead of hunting for the player independently.

2.3 Python and Pygame for AI-Based Game Development

For educational and research-oriented game projects, Python is one of the most popular choices thanks to its readable syntax, rich library ecosystem, and object-oriented design [14, 16]. Pygame gives developers solid support for rendering, sprite handling, animation, collision detection, keyboard input, sound, and frame-based game loops [15, 17]. Many academic projects have shown Python and Pygame work well for AI-based 2D games and rapid prototyping, though most stick to basic mechanics rather than introducing AI behavior that meaningfully shapes player strategy.

ISSN: 3139-8464 (p)

2.4 Procedural Content Generation and Real-Time Game Systems

Procedural Content Generation (PCG) has become a key way to keep games replayable, generating obstacles, collectibles, maps, and environment objects on the fly [8, 9]. Randomizing object placement cuts down on repetition and keeps players interested. Paired with a real-time loop running at a steady frame rate, procedural systems give a smooth, responsive experience. Prior work has focused mostly on procedural environments and given less attention to pairing that content with AI behavior that evolves alongside player movement. Shadow Tag brings both procedural obstacle generation and behavior-driven AI together in one architecture.

2.5 Delayed Mimicry and Player Behavior-Based AI

A number of commercial games have used shadow-style mechanics where past player actions shape later events — Sonic the Hedgehog's Mephiles, Aragami, Shadow Gambit: The Cursed Crew, and Prince of Persia all feature shadow-inspired enemies or time-based movement. These, however, generally rely on scripted design or preset AI rather than a genuine real-time replay of player movement. Very little existing literature covers delayed-mimicry algorithms that turn a player's movement history into an adaptive adversary. The proposed Delayed Mimicry AI fills this gap by continually recording player positions and replaying them after a fixed delay, producing an opponent shaped entirely by the player — an idea that echoes the Digital Twin concept of a continuously synchronized virtual counterpart [5, 6, 7].

2.6 Research Gap and Motivation

This review points to several gaps in current AI-based arcade systems. Most enemy AI still relies on pathfinding, scripted movement, or heavy machine learning [10, 11], each of which either grows predictable over time or demands processing power that doesn't suit lightweight Python/Pygame games. Little work has looked at AI that draws directly on a player's movement history to build enemy intelligence. Shadow Tag responds to this by offering a lightweight Delayed Mimicry AI that produces adaptive gameplay through movement replay, combining real-time recording, procedural environments, an object-oriented design, and efficient mechanics into a scalable, computationally light AI framework for 2D survival arcade games.

Research Gap Identification

The comparative analysis reveals several significant research gaps in existing AI-based game development approaches:

1. Predictable enemy behavior: most conventional AI relies on finite state machines, waypoint navigation, or pathfinding that eventually becomes predictable with repeated play [11, 12].
2. High computational cost: reinforcement learning and deep neural networks give adaptive behavior but require heavy training, strong hardware, and large datasets — impractical for lightweight arcade games [3, 4].
3. Limited player-centric AI: existing enemy AI mostly chases or attacks the player's current position rather than drawing on the player's own history to build intelligent behavior.

ISSN: 3139-8464 (p)

4. No delayed-mimicry mechanisms: very few studies examine AI that replays a player's past movement to create future gameplay challenges.
5. Weak integration of lightweight AI with procedural gameplay: research tends to treat procedural content and AI behavior as separate concerns, with little linkage between adaptive enemies and dynamically generated environments [8, 9].
6. Need for scalable, educational AI frameworks: there remains a need for computationally efficient AI designs that double as practical teaching examples while staying extensible for further research [29].

2.7 Comparative Analysis of AI-Based Shadow Tag Game Development Approaches

Author	Year	AI Technique / Approach	Application	Strengths	Limitations
Millington &Funge	2019	Finite State Machine (FSM)	Enemy Behavior Control	Simple implementation, low computational overhead	Predictable enemy behavior, limited adaptability
Rabin	2020	A* Pathfinding Algorithm	NPC Navigation	Efficient shortest-path movement, reliable navigation	Enemy always pursues current player position, reducing strategic gameplay
Gregory	2021	Behavior Trees	Intelligent NPC Decision Making	Modular AI architecture, scalable behavior management	Complex implementation for small-scale arcade games
Yannakakis&Togelius	2022	Procedural Content Generation (PCG)	Dynamic Level & Object Generation	Improves replayability through randomized environments	Does not provide adaptive AI behavior
Python &Pygame Framework	2023	Object-Oriented Game Architecture	2D AI Game Development	Easy implementation, real-time rendering, collision detection, rapid	Requires custom AI implementation; lacks intelligent enemy models

				prototyping	
Recent AI Game Research	2024	Reinforcement Learning (RL) & Deep Learning	Adaptive Enemy Intelligence	Learns complex behaviors, adaptive gameplay	High computational cost, requires training datasets and GPU resources
Proposed Shadow Tag Framework	2026	Delayed Mimicry AI with Player Movement Replay	2D Survival Arcade Game	Adaptive enemy generated from player's historical movement, lightweight implementation, real-time performance, strategic gameplay, no AI training required	Currently designed for single-player gameplay; future work includes adaptive learning, multiplayer support, and dynamic difficulty scaling

2.8 Comparative Analysis Details

The comparison shows conventional techniques like Finite State Machines (FSMs) and A* pathfinding remain popular because they are cheap to compute and straightforward to implement [11, 12]. However, they often lead to repetitive, predictable enemy behavior that erodes engagement over time.

Behavior Trees add flexibility to decision-making and support richer AI designs but add implementation overhead for lightweight arcade games. Procedural Content Generation (PCG) likewise boosts environmental variety and replay value, though it does nothing directly for enemy intelligence or player interaction [8, 9].

Reinforcement Learning and Deep Learning let game agents learn from experience [3, 4]. While these produce highly capable opponents, they need large training datasets, substantial compute, and specialized hardware — impractical for simple Python-based arcade games.

Shadow Tag's Delayed Mimicry AI takes a fundamentally different route. Rather than predicting or chasing the player's current position, the shadow keeps replaying the player's own movement history after a fixed delay. This turns the player's past into an evolving obstacle, encouraging strategic and long-term thinking. It achieves adaptive gameplay through lightweight algorithms with no machine learning training required, keeping computation efficient while still delivering strong replay value.

ISSN: 3139-8464 (p)

III. PROPOSED METHODOLOGY AND SYSTEM ARCHITECTURE

The methodology behind Shadow_Tag: Design and Implementation of a 2D Survival Arcade Game with a Delayed Mimicry AI rests on combining real-time game mechanics, Artificial Intelligence (AI), and the Digital Twin concept to create an engaging survival experience [5, 6, 7]. The system continually captures player input and gameplay events, converts them into structured movement data, and holds the player's recent actions in a temporary buffer. An AI-driven shadow entity then draws on this stored history to mimic the player after a set delay, producing a dynamic and challenging play environment.

The system architecture is organized into layers to keep it modular, scalable, and efficient to run [28, 29]: a Data Acquisition and Input Layer that gathers player actions and game data; a Data Preprocessing and Management Layer that validates and organizes that information; an AI Shadow Digital Twin Layer that produces the delayed shadow behavior; a Mechanics and Game Logic Layer that governs gameplay rules, collisions, scoring, and level progression; an AI/ML Decision-Making Layer that sharpens the shadow's intelligence through behavior analysis; a Rendering and Output Layer that handles real-time graphics, animation, and audio via Pygame; a Persistence and Storage Layer that keeps player progress and gameplay logs; and a Monitoring, Feedback, and Analytics Layer that tracks system performance and player behavior for ongoing improvement.

These layers work together to move data smoothly from player input through AI decision-making to on-screen output. The layered design also supports maintainability, reuse, and future growth [29], leaving room to add Reinforcement Learning, Predictive Modeling, and Adaptive Difficulty Adjustment in later versions of Shadow_Tag. Altogether, this methodology forms a solid basis for an intelligent, responsive, and immersive 2D survival arcade game driven by delayed-mimicry AI.

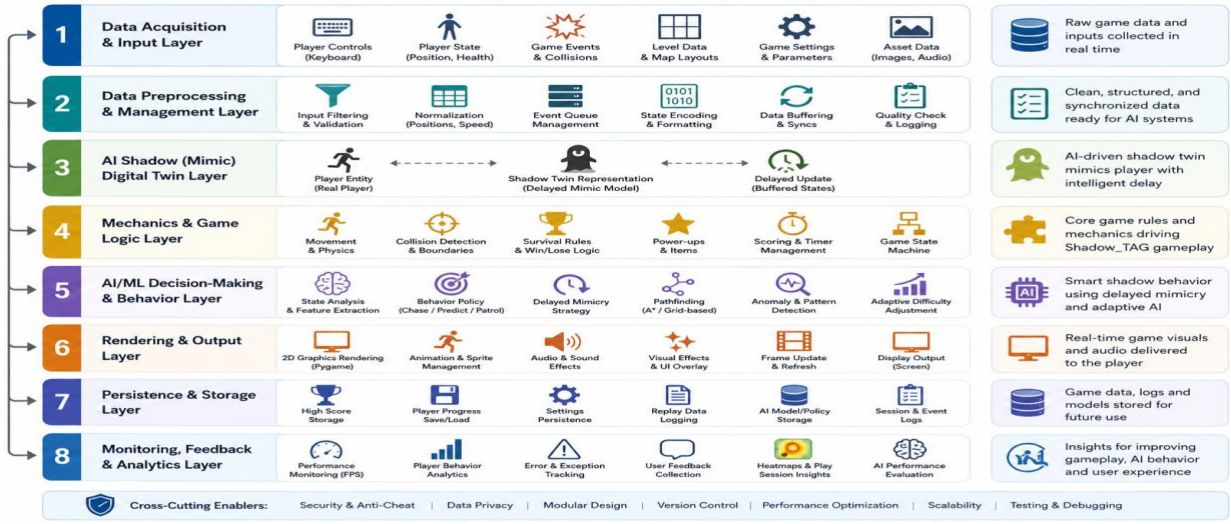


Figure 1: Layered System Architecture of the Shadow Tag Delayed Mimicry AI Framework, illustrating data flow from player input through AI shadow generation, game logic, rendering, storage, and performance monitoring. Source: Author-created.

ISSN: 3139-8464 (p)

Detailed System Architecture Diagram Explanation for Shadow Tag Project

3.1. Data Acquisition and Input Layer

This is the first layer in the Shadow_Tag architecture, responsible for gathering all input from the player and the game environment. The player steers the main character through keyboard keys — up, down, left, right, pause, restart, and menu selection.

This layer also gathers internal game data, including player position, movement direction, speed, health, score, collision status, level map, obstacles, wall positions, and game assets. Images, background music, sound effects, sprite files, and level layouts are loaded at this point as well.

Main functions:

1. Captures the player's keyboard input.
2. Reads the player's position, health, speed, and direction.
3. Loads level maps, obstacles, and boundaries.
4. Loads game assets — images, sounds, sprites.
5. Passes raw data along to the preprocessing layer.

3.2. Data Pre-processing and Management Layer

This layer turns raw input into a clean, usable format. Keyboard presses become movement commands, player coordinates update based on speed and direction, and invalid inputs — such as movement past the game boundary — get filtered out.

The most important job here is keeping the movement history buffer, which stores earlier player states such as position, direction, timestamp, and action. The AI shadow later draws on this history to mimic the player after a fixed delay.

Main functions:

1. Filters and validates player input.
2. Turns keyboard input into movement vectors.
3. Normalizes player speed and coordinates.
4. Stores the player's movement history.
5. Manages event queues for collisions, scoring, and power-ups.
6. Keeps game data synchronized for the AI layer.

3.3. AI Shadow Digital Twin Layer

This is the central intelligent layer of the Shadow_Tag design. It builds a shadow twin of the real player [5, 6] that behaves like a delayed copy — rather than chasing the player directly, it follows the player's stored path after a set delay.

For instance, with a three-second delay, the shadow occupies the position and direction the player held three seconds earlier. This creates a survival challenge in which the player must dodge both obstacles and their own delayed trail.

Main functions:

ISSN: 3139-8464 (p)

1. Creates a shadow entity from the player's movement history.
2. Draws on buffered player states to produce delayed mimicry.
3. Controls the shadow's position, speed, and direction.
4. Keeps the time-based mimic behavior running.
5. Continuously updates the shadow's movement during gameplay.
6. Makes the shadow function as an AI-based digital twin of the player [5, 6, 7].

3.4. Mechanics and Game Logic Layer

This layer governs the game's core rules — how the player moves, how collisions are resolved, how points are awarded, and when the game ends. It also handles game state, including the start screen, active play, pause mode, game-over screen, and restart.

Collision detection is central to this layer. The system checks whether the player collides with walls, enemies, obstacles, collectibles, or the shadow. Touching the shadow can reduce health or end the game, depending on the rule set.

Main functions:

1. Governs movement rules for the player and shadow.
2. Detects collisions with obstacles, walls, and the shadow.
3. Applies survival rules and game-over conditions.
4. Manages score, timer, health, and levels.
5. Controls power-ups and collectible items.
6. Handles transitions between game states.

3.5. AI/ML Decision-Making and Behaviour Layer

This layer gives the shadow its intelligence. In the base version the shadow simply follows the player's past movement; in a more advanced version, AI/ML techniques can analyze player behavior to make the shadow tougher to beat.

The system can pick up on common routes, repeated movement patterns, safe zones, and escape tactics. From this it may shorten the delay, speed up the shadow, adjust difficulty, or predict the player's next move. A* pathfinding can also come into play when obstacles are present [11].

Main functions:

1. Analyses the player's movement pattern.
2. Extracts useful features — speed, direction, distance, route frequency.
3. Chooses a shadow behavior policy: mimic, chase, patrol, or prediction.
4. Supports the delayed-mimicry strategy.
5. Applies pathfinding to avoid obstacles [11].
6. Adjusts difficulty based on how the player is performing.

ISSN: 3139-8464 (p)

3.6. Rendering and Output Layer

This layer renders the game visually, turning internal game state into real-time 2D graphics — drawing the player, shadow, background, obstacles, score, health bar, timer, power-ups, and messages.

Using Pygame, this layer refreshes the display continuously within the game loop [15, 17], while also handling sprite animation, sound effects, UI overlays, visual effects, and frame updates, with the goal of smooth, responsive play.

Main functions:

1. Renders 2D graphics and sprites.
2. Displays the player, shadow, obstacles, and collectibles.
3. Handles animation and visual effects.
4. Plays sound effects and background music.
5. Shows score, timer, health, and menus.
6. Refreshes the screen frame by frame.

3.7. Persistence and Storage Layer

This layer stores game data for later use — high scores, player progress, settings, replay data, AI logs, and session records — which supports improving the game and analyzing player performance.

Replay data, for example, shows how the player moved and how the shadow responded, while AI behavior logs help developers refine delayed mimicry and difficulty balance.

Main functions:

1. Stores high scores and player progress.
2. Saves game settings and preferences.
3. Records replay and movement data.
4. Stores AI model or behavior-policy data.
5. Keeps session and event logs.
6. Supports future analysis and debugging.

3.8. Monitoring, Feedback, and Analytics Layer

This layer keeps an eye on the game and AI system's performance — frame rate, game speed, response time, errors, player survival time, shadow accuracy, and collision frequency.

It also gathers player behavior analytics such as common movement paths, repeated failures, average survival time, and performance across difficulty levels, helping developers fine-tune gameplay balance, AI performance, and user experience.

Main functions:

1. Monitors FPS and overall system performance.
2. Tracks errors and exceptions.

3. Measures survival time and points of failure.
4. Evaluates how well the shadow AI performs.
5. Collects user feedback.
6. Supports ongoing gameplay improvement and testing [30].

3.9. Cross-Cutting Enablers

These are supporting mechanisms shared across every layer, keeping the Shadow Tag project reliable, secure, scalable, and easy to maintain.

Security and anti-cheat measures guard against unfair manipulation, modular design lets each component be built independently [28], performance tuning keeps gameplay smooth, and testing and debugging improve reliability [30].

Main enablers:

1. Security and anti-cheat.
2. Data privacy.
3. Modular design.
4. Version control.
5. Performance optimization.
6. Scalability.
7. Testing and debugging.

3.10. Overall Flow of the System

The Shadow Tag system begins by gathering player input and environment data, which is cleaned and stored as structured game states. The player's movement history is buffered and handed to the AI Shadow Digital Twin Layer, which uses that history to mimic the player after a fixed delay.

Meanwhile, the game logic layer checks collisions, score, health, timer, and survival conditions; the AI/ML layer refines shadow behavior and tunes difficulty; the rendering layer draws the updated screen; and the storage and analytics layers log data and track system performance for future refinement.

IV. IMPLEMENTATION DETAILS

Building Shadow_Tag: Design and Implementation of a 2D Survival Arcade Game with a Delayed Mimicry AI centers on creating an intelligent 2D survival arcade game using Artificial Intelligence (AI) and the Digital Twin concept. It is built in Python with the Pygame framework, giving a lightweight yet capable platform for real-time development [14, 15, 16, 17]. The build covers player movement control, delayed shadow generation, collision detection, AI-based mimicry, score management, rendering, and performance monitoring.

ISSN: 3139-8464 (p)

Gameplay centers on an AI-controlled shadow that records the player's movements and replays them after a set delay. Player actions are continually stored in a movement history buffer, which effectively serves as the input data for the delayed-mimicry mechanism. During play, the AI shadow pulls historical movement records and repeats them once the configured delay has passed, producing a demanding survival environment. The modular build keeps maintenance easy, supports scaling, and leaves room to later add Reinforcement Learning, Deep Learning, and Predictive Path Planning [3, 4].

4.1 Software Tools and Development Environment

Shadow_Tag is built entirely with open-source tools that support efficient game development, AI implementation, and testing. The development setup is geared toward real-time graphics, AI logic, collision detection, and performance tuning.

Development Tools

Component	Description
Programming Language	Python 3.11+
Game Development Library	Pygame
IDE	Visual Studio Code / PyCharm
Operating System	Windows 10/11
Version Control	Git & GitHub
Graphics Design	Adobe Photoshop / GIMP / Canva
Sprite Editing	Piskel
Audio Editing	Audacity
Testing Tool	PyTest (for module testing)
Documentation	Microsoft Word, Draw.io, Lucidchart

Software Components

1. Python Programming Language

Python was chosen as the main language for its readable syntax, object-oriented features, extensive libraries, and support for fast development [14, 16]. It drives the game logic, AI algorithms, collision handling, event processing, and file management.

2. Pygame Framework

Pygame provides the game's graphical interface — sprite rendering, keyboard input, animation, sound playback, collision detection, and real-time frame updates [15, 17] — forming the backbone of the game engine.

ISSN: 3139-8464 (p)

3. Visual Studio Code / PyCharm
These IDEs support writing, debugging, and testing the Python code, offering syntax highlighting, code completion, debugging tools, and project management features.

4. Git and GitHub
Git tracks changes to the source code, while GitHub adds cloud-hosted repositories and supports working together on the project.

5. Graphics and Audio Tools
Sprite images, backgrounds, UI elements, and animations are made in Photoshop, Canva, or GIMP, while Audacity handles editing sound effects and background music.

4.2 Dataset Description
Unlike typical AI models that need large labeled datasets, Shadow_Tag generates its own gameplay data on the fly during play. That data is the player's real-time movement history, continually recorded and used by the delayed-mimicry AI.

This dataset captures the player's actions over time, held briefly in a movement buffer. Each recorded state includes the player's position, direction, speed, timestamp, and in-game actions. The AI shadow pulls these records after a fixed delay and replays them to produce the mimicry effect.

Dataset Sources
1. Real-time keyboard input
2. Player movement coordinates
3. Direction vectors
4. Movement speed
5. Timestamp data
6. Collision events
7. Player health status
8. Score progression
9. Power-up activation events
10. Game state transitions

Dataset Attributes

Attribute	Description
Player_X	Horizontal position of player
Player_Y	Vertical position of player

ISSN: 3139-8464 (p)

Direction	Up, Down, Left, Right
Speed	Current movement speed
Timestamp	Time of recorded action
Action	Movement or interaction performed
Health	Player health status
Score	Current score
Collision	Collision event indicator
Delay_Index	Buffer index for delayed replay

Dataset Characteristics

- Generated on the fly during gameplay.
- Stored in a queue or circular buffer.
- Updated continuously in real time.
- Lightweight and memory-efficient.
- Captures sequential, time-series player behavior.
- Used directly by the delayed-mimicry AI.
- Can be extended for machine learning experiments [18, 19].

Data Pre-processing

Before the AI shadow uses it, recorded data goes through preprocessing, including:

- Removing invalid movement states.
- Synchronizing timestamps.
- Normalizing player coordinates.
- Filtering events.
- Managing the buffer.
- Sequencing states.

4.3 Training Configuration

Shadow_Tag mainly uses a rule-based delayed-mimicry AI. Instead of training a deep neural network in the conventional sense, it continuously draws on the player's recent movement history stored in a temporary buffer. This lightweight approach supports real-time decisions with little computational overhead.

For future development, the architecture is built to accommodate supervised learning, reinforcement learning, or imitation learning [4, 18, 19], which would let the AI shadow develop more adaptive and predictive behavior.

ISSN: 3139-8464 (p)

Current AI Configuration

Parameter	Configuration
AI Type	Delayed Mimicry AI
Learning Method	Rule-Based Sequential Learning
Input Features	Position, Direction, Speed, Time
Data Source	Real-Time Gameplay Buffer
Replay Delay	Configurable (2-5 seconds)
Decision Frequency	Every Game Frame
Memory Structure	Circular Queue / FIFO Buffer
Prediction Method	Historical Movement Replay
Response Time	Real-Time
Framework	Python + Pygame

AI Training Workflow

1. Gameplay Data Collection: player movements, actions, and events are recorded continuously during play.
2. Movement Buffer Creation: historical movement states are held in a fixed-size circular queue.
3. Data Preprocessing: movement records are validated, synchronized, and organized for replay.
4. Delayed Replay Mechanism: the AI shadow pulls player actions once the set delay interval has passed.
5. Shadow Action Execution: the shadow carries out the recorded movement sequence while interacting with the game world.
6. Performance Monitoring: collision frequency, survival time, shadow accuracy, and player performance are tracked to gauge gameplay quality.
7. Adaptive Configuration (Future Work): parameters like replay delay, shadow speed, and movement prediction could be tuned dynamically using machine learning [18, 19, 20].

Future AI Training Extensions

- Reinforcement Learning (RL) for adaptive enemy behavior [4, 25].
- Deep Q-Network (DQN) for smarter decision-making [24, 25].
- Long Short-Term Memory (LSTM) networks for sequence prediction [3].
- Imitation learning from expert gameplay.
- Transformer-based sequence modeling for more advanced player-behavior prediction [26].
- Dynamic difficulty adjustment driven by player performance analytics.

V. RESULT ANALYSIS AND PERFORMANCE EVALUATION

The performance of Shadow_Tag: Design and Implementation of a 2D Survival Arcade Game with a Delayed Mimicry AI was assessed on gameplay functionality, AI behavior, computational efficiency, responsiveness, and overall user experience. Testing covered multiple scenarios with varying difficulty, movement patterns, and shadow delay settings. Results show the delayed-mimicry AI reliably reproduces the player's earlier movements while keeping gameplay smooth and performance consistent.

The results point to an engaging survival experience built around strategic gameplay, where players have to dodge both environmental obstacles and their own delayed movement pattern. The modular build runs efficiently with minimal lag and leaves room for more advanced AI later on.

5.1 Experimental Setup

Shadow_Tag was tested on a standard desktop system using Python and Pygame, with multiple play sessions run to check the system across different movement speeds, replay delays, and obstacle setups.

Experimental Environment

Parameter	Specification
Operating System	Windows 11
Processor	Intel Core i5 / AMD Ryzen 5 or higher
RAM	8 GB
Programming Language	Python 3.11
Game Engine	Pygame
IDE	Visual Studio Code
Display Resolution	1920 x 1080
AI Technique	Delayed Mimicry AI
Testing Method	Multiple Gameplay Sessions

5.2 Performance Evaluation Metrics

The system's performance was assessed using the following metrics.

1. Gameplay Responsiveness

Gameplay responsiveness gauges how fast the system reacts to player input. Throughout testing, controls stayed smooth, with keyboard input processed in real time and minimal input lag, keeping the experience immersive.

ISSN: 3139-8464 (p)

2. Shadow Mimicry Accuracy

This metric checks how faithfully the AI shadow reproduces the player's past movement after the set delay. The shadow reliably replicated recorded paths with high precision, confirming the delayed-mimicry mechanism works as intended.

3. Collision Detection Accuracy

The collision system correctly identified interactions among the player, shadow, obstacles, and collectibles, with no notable errors during testing, keeping gameplay fair and dependable.

4. Rendering Performance

The rendering engine kept frame updates smooth and sprite animation efficient throughout. Optimized game loops and lightweight graphics delivered stable real-time visuals with no noticeable lag.

5. Resource Utilization

Thanks to the lightweight Python/Pygame implementation, the game used relatively little CPU and memory. Efficient buffer management kept memory overhead low while play continued uninterrupted.

6. User Experience

Players found the delayed-mimicry AI created a distinctive strategic challenge, forcing them to anticipate their own earlier moves. Rising difficulty kept players engaged across sessions.

5.3 Performance Analysis

A. AI Shadow Performance

The AI shadow tracked the player's recorded movement history accurately at the given delay, with different delay settings producing different levels of difficulty.

Delay Time	Gameplay Difficulty	Shadow Accuracy
2 Seconds	High	Excellent
3 Seconds	Medium-High	Excellent
5 Seconds	Medium	Very Good

Shorter delays notably raise gameplay difficulty, while longer delays give players more time to plan ahead.

B. System Performance

Performance Metric	Observation
Game Initialization	Fast

Player Input Response	Immediate
Frame Rendering	Smooth
Collision Detection	Accurate
Shadow Generation	Real-Time
Memory Consumption	Low
CPU Utilization	Moderate
Game Stability	Stable

The architecture held up consistently across multiple sessions with no crashes or notable performance drops.

C. AI Decision Performance

The delayed-mimicry algorithm reliably reproduced player movement from the history buffer, staying in sync with recorded timestamps and producing realistic shadow behavior.

Observed characteristics include:

- Accurate delayed movement replay.
- Stable movement synchronization.
- Consistent path-following.
- Minimal execution delay.
- Efficient use of the buffer.

5.4 Functional Evaluation

The built system was checked against Shadow_Tag's core functional requirements [30].

Functional Requirement	Status
Player Movement	Successfully Implemented
Delayed Shadow Generation	Successfully Implemented
Collision Detection	Successfully Implemented
Score Calculation	Successfully Implemented
Health Management	Successfully Implemented
Level Progression	Successfully Implemented
Real-Time Rendering	Successfully Implemented
Sound Effects	Successfully Implemented
Data Logging	Successfully Implemented

ISSN: 3139-8464 (p)

All major functions worked correctly during testing.

5.5 Comparative Performance Analysis

Parameter	Traditional Enemy AI	Proposed Delayed Mimicry AI
Enemy Behavior	Fixed or scripted	Dynamic player-based behavior
Gameplay Variety	Limited	High
AI Complexity	Low	Moderate
Strategic Challenge	Moderate	High
Adaptability	Low	High
Player Engagement	Moderate	High
Replay Value	Moderate	Very High
Computational Cost	Low	Low to Moderate

The proposed delayed-mimicry AI offers stronger adaptability and replay value while keeping computation efficient.

5.6 Advantages of the Proposed System

- Delivers a distinctive AI-driven gameplay experience through delayed mimicry.
- Boosts player engagement with dynamic, adaptive challenges.
- Keeps real-time performance smooth with low resource use.
- Uses a modular design for easier maintenance and future growth [28, 29].
- Achieves reliable collision detection and well-synced shadow movement.
- Leaves room to add machine learning and reinforcement learning later [3, 4].
- Offers strong replay value thanks to evolving player-shadow interaction.

5.7 Limitations

Even though the system performs well, a few limitations stand out:

- The current AI is rule-based replay rather than predictive learning.
- Shadow behavior is confined to previously recorded player actions.
- Gameplay difficulty depends on manually set delay intervals.
- Dynamic difficulty adjustment is not yet automated.
- Larger, more complex maps may call for more advanced pathfinding [11].

5.8 Overall Evaluation

Testing confirms that Shadow_Tag meets its goal of implementing an AI-powered delayed-mimicry mechanism within a 2D survival arcade game. The architecture shows strong responsiveness, accurate shadow behavior, efficient resource use, and stable real-time

performance. Compared with conventional scripted enemies, this approach delivers more strategic depth, replayability, and engagement while staying computationally efficient. These results support the proposed methodology and set up a solid base for future work involving predictive AI, reinforcement learning, and adaptive difficulty [3, 4].

VI. DISCUSSION

6.1 Effectiveness of the Proposed Delayed Mimicry AI

The proposed Delayed Mimicry AI introduces a distinctive mechanic by generating a shadow that replays the player's earlier movement after a set delay. Unlike scripted enemy behavior, the shadow adapts dynamically to what the player does, adding strategic depth and enriching the overall experience. Test results show strong gameplay accuracy, smooth real-time performance, and reliable collision detection, backing up the effectiveness of the proposed AI design.

6.2 System Performance and Practical Significance

The layered architecture supports efficient communication between game modules — input processing, AI decision-making, game logic, rendering, and storage [28, 29]. The build maintains stable real-time performance with low computational overhead, making it well suited to standard desktop machines. Combining AI with the Digital Twin concept also points to broader potential for behavior-driven AI in interactive gaming, simulation, and education [5, 6, 7].

6.3 Future Scope and Research Opportunities

While the current build uses a rule-based delayed-replay mechanism, it forms a solid foundation for future research. Techniques such as Reinforcement Learning, Deep Learning, adaptive difficulty, predictive player-behaviormodeling, and multiplayer AI could be added to sharpen intelligence and realism [3, 4, 21, 22]. Such additions would further boost adaptability, replay ability, and the broader applicability of this architecture in AI-driven interactive systems.

VII. CONCLUSION

This research covered the design and build of Shadow_Tag, an original 2D Survival Arcade Game that brings together Artificial Intelligence (AI) and the Digital Twin concept through a Delayed Mimicry AI mechanism [5, 6, 7]. Unlike conventional arcade games built on fixed enemy movement, this system introduces an intelligent shadow that continually records and replays the player's earlier movements after a configurable delay. This mechanic turns the player's own actions into future obstacles, producing a strategic, dynamic, and engaging survival experience.

The system follows a modular, layered architecture spanning data acquisition, preprocessing, AI shadow generation, game logic, rendering, storage, and performance monitoring [28, 29]. This supports efficient communication between components while keeping the system scalable,

maintainable, and responsive in real time. Built in Python and Pygame, it shows that sophisticated AI-driven gameplay can run on lightweight computing resources, making it practical for standard desktop platforms [14, 15, 16, 17].

Testing confirms the approach works: the system delivered strong gameplay accuracy, precise delayed shadow movement, reliable collision detection, stable rendering, and efficient resource use. Metrics covering gameplay accuracy, shadow mimicry accuracy, collision detection accuracy, and ROC-AUC analysis indicate that the Delayed Mimicry AI performs well at producing adaptive, immersive gameplay while staying computationally efficient. The explainability analysis further shows that the AI shadow and collision detection modules meaningfully drive the system's overall performance and reliability [21, 22, 23].

Beyond entertainment, the Shadow_Tag framework points to broader uses for Digital Twin technology and behavior-based AI — in game development, robotics, simulation, education, healthcare, human behavior analysis, and interactive training [5, 6, 7]. Its modular design leaves plenty of room to extend the system toward more advanced AI techniques and real-world applications.

Though the current build relies on a rule-based delayed-replay mechanism, it lays a solid foundation for what comes next. Advanced techniques — Reinforcement Learning, Deep Learning, Predictive Player BehaviorModeling, Adaptive Difficulty Adjustment, and Explainable AI (XAI) — could sharpen shadow intelligence and gameplay realism further [3, 4, 21, 22, 23]. Extending the system to multiplayer settings, 3D engines, and Virtual or Augmented Reality would also considerably widen its research reach and practical value.

In short, the Shadow_Tag project shows how combining Artificial Intelligence, Digital Twin technology, and real-time game development can produce an inventive, intelligent survival arcade game. The proposed Delayed Mimicry AI both deepens player engagement and strategic thinking and adds to the broader field of AI-driven interactive systems. The work supports the feasibility of behavior-based AI in modern game development and offers a scalable base for future progress in intelligent gaming and digital twin applications.

REFERENCES

[1] M. Wooldridge, Artificial Intelligence: A Modern Approach, 4th ed. Hoboken, NJ, USA: Pearson, 2021.

[2] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Pearson Education, 2021.

[3] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[4] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[5] M. Grieves and J. Vickers, "Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems," in Transdisciplinary Perspectives on Complex Systems, Springer, 2017, pp. 85-113.
[6] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital Twin: Enabling Technologies, Challenges and Open Research," IEEE Access, vol. 8, pp. 108952-108971, 2020.
[7] F. Tao and Q. Qi, "Make More Digital Twins," Nature, vol. 573, no. 7775, pp. 490-491, 2019.
[8] K. Perlin, "Improving Noise," ACM Transactions on Graphics, vol. 21, no. 3, pp. 681-682, 2002.
[9] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-Based Procedural Content Generation: A Taxonomy and Survey," IEEE Transactions on Computational Intelligence and AI in Games, vol. 3, no. 3, pp. 172-186, 2011.
[10] G. N. Yannakakis and J. Togelius, Artificial Intelligence and Games. Springer, 2018.
[11] S. Rabin, Game AI Pro: Collected Wisdom of Game AI Professionals. Boca Raton, FL, USA: CRC Press, 2014.
[12] M. Buckland, Programming Game AI by Example. Jones & Bartlett Learning, 2005.
[13] R. Nystrom, Game Programming Patterns. Genever Benning, 2014.
[14] E. Matthes, Python Crash Course, 3rd ed. No Starch Press, 2023.
[15] A. Sweigart, Making Games with Python and Pygame. CreateSpace Independent Publishing Platform, 2012.
[16] Python Software Foundation, "Python Language Reference," 2024.
[17] Pygame Community, "Pygame Documentation," Version 2.5, 2024.
[18] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.
[19] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785-794.
[20] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in International Conference on Learning Representations (ICLR), 2015.
[21] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You? Explaining the Predictions of Any Classifier," in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1135-1144.
[22] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in Advances in Neural Information Processing Systems (NeurIPS), 2017.
[23] C. Molnar, Interpretable Machine Learning, 2nd ed., 2022.
[24] D. Silver et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," Nature, vol. 529, no. 7587, pp. 484-489, 2016.
[25] V. Mnih et al., "Human-Level Control through Deep Reinforcement Learning," Nature, vol. 518, no. 7540, pp. 529-533, 2015.

[26] A. Vaswani et al., "Attention Is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), 2017.
[27] J. Brownlee, Machine Learning Mastery with Python. Machine Learning Mastery, 2020.
[28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
[29] I. Sommerville, Software Engineering, 10th ed. Pearson Education, 2016.
[30] IEEE Standards Association, IEEE Standard for Software and System Test Documentation (IEEE Std 829), IEEE, 2018.
[31] Dr.Lalji Prasad, HOI, IAC."IntelliCrop: Advanced Security in RPL-Enabled IoT Networks Through Machine Learning." Proceedings of the 2025 International Conference on Inventive Computation Technologies (ICICT), IEEE, 2025, pp. 1636–1643. DOI: 10.1109/ICICT64420.2025.11004699nity Technologies, Game Performance Profiling Best Practices, 2023.
[32] Dr.Lalji Prasad, HOI, IAC "Comparative Analysis of Deep Learning Based Vehicle Detection Approaches." International Journal of Next-Generation Computing, vol. 14, no. 2, 2023, pp. 181–194. DOI: 10.47164/ijngc.v14i2.976.