RV College of Engineering®

Mysore Road, RV Vidyaniketan Post,Bengaluru - 560059, Karnataka, India

AI-Based Cyber Defense Game for Network Security Education

AIML-PROJECT REPORT

Submitted by

Keerthi M **1RV23IS061**

Mahika Marpuri **1RV23IS066**

Under the guidance of

Dr. Merin Meleet
Associate Professor
Dept. of ISE
RV College of Engineering

In partial fulfillment for the award of degree of

Bachelor of Engineering

in

Information Science and Engineering

2025-2026

RV COLLEGE OF ENGINEERING®, BENGALURU - 560059

(Autonomous Institution Affiliated to VTU, Belagavi) DEPARTMENT OF

INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work titled **"AI-Based Cyber Defense Game for Network Security Education"** is carried out by **Keethi M (1RV23IS066) and Mahika (1RV23IS066),** who are bonafide student of R.V College of Engineering, Bangalore, in partial fulfillment for the award of degree of **Bachelor of Engineering** in **Information Science and engineering** of the Visvesvaraya Technological University, Belgaum during the year **2025-26**. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project work as a part of the course "Artificial Intelligence and Machine Learning" prescribed by the institution for the said degree.

Faculty in Charge **Head of the Department**

External Evaluation

Name of Examiners **Signature with date**

1

2

RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,Bengaluru - 560059, Karnataka, India

DECLARATION

We, **Keerthi M and Mahika Marpuri** student of fifth semester B.E., Department of Information Science and Engineering, RV College of Engineering, Bengaluru, hereby declare that the project titled "**AI-Based Cyber Defense Game for Network Security Education**" has been carried out by us and submitted in partial fulfilment for the award of degree of **Bachelor of Engineering** in **Information Science and Engineering** during the year 2025-2026

Further we declare that the content of the dissertation has not been submitted previously by anybody for the award of any degree or diploma to any other university.

I also declare that any Intellectual property rights generated out of this project carried out at RVCE will be the property of RV College of Engineering, Bengaluru and I will be among the author of the same.

_____

Place: Bengaluru

Date:

Signature

ACKNOWLEDGEMENTS

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped me in carrying out this project. We would like to take this opportunity to thank them all.

We are thankful to our faculty, Dr. Merin Meleet, and B K Srinivas, Dept. of ISE for their wholehearted support, suggestions and advice during this work.

Our sincere thanks to Dr. G S Mamatha, Professor and Head, Department of Information Science and Engineering, RVCE for his support and encouragement.

We express sincere gratitude to our beloved Principal, Dr. K. N. Subramanya for his appreciation towards this work.

We thank all the teaching staff and technical staff of the Information Science and Engineering department, RVCE for their help.

ABSTRACT

Cybersecurity continues to be one of the most pressing domains in the modern digital era, yet foundational concepts such as network vulnerabilities, intrusion strategies, and countermeasures often remain abstract to beginners. Network Defender is an AI-powered educational tower-defense simulation designed to simplify cybersecurity awareness by modeling network traffic as attack packets attempting to breach a core server. The player deploys defensive mechanisms such as Firewalls, Intrusion Detection Systems (IDS), and Honeypots across a custom-generated network topology to mitigate threats. Through interactive gameplay and real-time feedback, the system promotes intuitive learning of cyber defense strategies while maintaining engaging gameplay.

The methodology combines a Pygame-based simulation framework with Adaptive AI that analyzes player defense patterns and dynamically evolves attack traffic. Various packet types such as basic malware, fast-moving intrusions, tank-type brute-force attacks, and encrypted stealth infiltrations challenge the user to diversify their defensive approach. Legitimate packets are included to emphasize real-world network filtering accuracy and minimize false positives. Additionally, the game introduces budgeting and placement constraints, forcing strategic trade-offs similar to cybersecurity resource allocation encountered in real systems.

Results demonstrate that users learn cybersecurity concepts more effectively when exposed to interactive threat-response cycles rather than static theory. Gameplay analytics and success rate tracking support both user performance evaluation and automated strategy recommendations. The project proves that gamified learning with adaptive AI can significantly enhance understanding of network security principles including traffic inspection, intrusion detection, resource allocation, and attack surface minimization.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API - Application Programming Interface

CI/CD - Continuous Integration / Continuous Deployment

CPU - Central Processing Unit

DOM - Document Object Model

DTO - Data Transfer Object

gRPC - google Remote Procedure Call

GPU - Graphics Processing Unit

HTTP/2 - Hypertext Transfer Protocol Version 2

JSON - JavaScript Object Notation

ML - Machine Learning

QPS - Queries Per Second

TMS - Translation Management System

TTL - Time To Live

Octavius - In-house ML engine for content ranking

Alchemist - In-house experimentation engine

ARMOUR - In-house content moderation system

Caffeine - High-performance Java in-memory cache

CHAPTER 1

INTRODUCTION

Digital infrastructures today are continuously targeted by cyber threats ranging from malware injections and DDoS attacks to encrypted intrusions. Traditional cybersecurity education often relies heavily on theoretical lectures and memorization of concepts without providing intuitive visualization of how attacks propagate through networks and how defenses interact in real time. This creates a gap between conceptual understanding and practical capability.

To address this gap, Network Defender transforms cybersecurity fundamentals into an interactive learning experience. The system visualizes network nodes as connection points, attack packets as adversarial agents, and defensive tools as deployable cybersecurity mechanisms. The challenge for the player is to prevent malicious traffic from reaching the core server under constrained resources using optimal placement and timing of defense towers. As levels progress, the complexity of attacks increases, encouraging users to learn security layering, prioritization, and budgeting.

1.1 Terminology

Term	Meaning
Packet	Data traveling across the network (legitimate or malicious)
Firewall	Defensive tower that damages harmful packets
IDS	Intrusion Detection System; slows and analyzes malicious packets
Honeypot	Decoy node that attracts malicious packets
Attack Vector	Method or pattern used by a malicious packet to infiltrate the network
Core Server	Final node that must not be breached
Adaptive AI	System that learns from user defenses and evolves packet strategies

1.2 Scope and Relevance

The project focuses on creating a gamified platform that visually simulates cyber-attacks and defensive mechanisms in a simplified network environment. Its scope includes real-time visualization of packets, implementation of core cybersecurity tools such as firewalls, IDS, and honeypots, and the use of adaptive AI to evolve attack patterns based on player behaviour.

The system is highly relevant for cybersecurity learners, engineering students, and beginners who struggle to understand abstract concepts like intrusion, packet filtering, and network defense. By integrating gameplay with learning, the project enhances user engagement, improves conceptual clarity, and provides a practical, interactive approach to cyber awareness training.

1.3 Motivation

Cybersecurity fundamentals are difficult for learners because threats are abstract and invisible. Gamification bridges this barrier by:

- Revealing cyberattacks visually
- Allowing experimentation with defense strategies
- Demonstrating consequences of decisions immediately

1.4 Problem Statement

Conventional tower defense games rely on static or pre-scripted enemy behaviors that do not evolve based on player strategies. This leads to repetitive, predictable gameplay and fails to represent real-world network security challenges, where attackers continuously adapt.

The primary problem addressed in this project is to design a game AI capable of learning and adapting dynamically, both during and across gameplay sessions, using reinforcement learning for real-time adaptation and supervised learning for long-term improvement. Additionally, incorporating dynamic network topologies enhances realism by simulating the constantly changing structure of real networks.

1.5 Objectives

1) Design a tower defense game centered around core network security principles.
2) Implement reinforcement learning (RL) for adaptive in-game enemy behavior based on live player responses.
3) Collect gameplay data and use supervised learning for offline model retraining to refine AI strategies.
4) Integrate dynamic network topologies to replicate evolving network conditions.
5) Provide visual and statistical feedback to players, showing how the AI has adapted and offering strategic improvement tips.
6) Create an intuitive and educational gaming platform that effectively combines AIML techniques and cybersecurity awareness.

1.6 Summary

The Introduction chapter explains the basic concepts of network security, describes why cybersecurity education needs interactive tools, and discusses how traditional theoretical methods make it difficult for beginners to visualize attacks and defenses. It presents the motivation behind creating a simulation-based learning platform and clearly defines the core problem: the absence of simple, accessible systems that demonstrate real-time packet flow and defensive mechanisms. The chapter also outlines the project's main objectives, such as simulating different attack types, integrating defensive tools like firewalls and IDS, and enabling strategy-based learning. Overall, it provides the background context, the need for the project, and the goals that guide the rest of the report.

CHAPTER 2

LITERATURE SURVEY

2.1 Literature Review

1. Nguyen et al., 2024 - Hybrid AI Framework for Game Adaptation
Nguyen et al. (2024) proposed a hybrid AI framework that integrates real-time reinforcement learning with offline supervised learning to enable adaptive behaviour in strategic game environments. Their work highlighted the potential of combining multiple AI models to improve decision-making and responsiveness in gameplay. However, the study remained mostly theoretical, with minimal implementation and limited experimental validation, leaving a gap in demonstrating how such hybrid AI can be applied to real interactive systems like cybersecurity simulations.

2. Lee et al., 2023 - Dynamic Difficulty Adjustment Using Reinforcement Learning
Lee et al. (2023) explored how reinforcement learning can automatically adjust game difficulty to match a player's skill level, significantly improving engagement and game balance. Their findings confirmed that RL can effectively learn optimal difficulty settings through player interaction. Despite its value, the work focused solely on difficulty balancing and did not explore more complex domains such as cybersecurity scenarios, behavioural modelling, or adaptive enemy strategies.

3. Singh & Rao, 2023 - Dynamic Network Simulation for Defense Training
Singh and Rao (2023) presented a dynamic network simulation system aimed at improving cyber-defense training by using variable network layouts to replicate realistic attack environments. Their work emphasized the importance of scenario diversity and network topology changes. However, the platform lacked interactive game elements, user participation, and AI/ML-based enemy modelling, limiting its applicability for real-time educational gameplay.

4. Patel et al., 2023 - Gamified Cybersecurity Training with AI-Driven Scenarios
Patel et al. (2023) demonstrated that integrating gamification with AI-driven cybersecurity scenarios significantly enhances learner engagement and understanding. Their system used AI to create structured attack scenarios suitable for beginner training. The main limitation was that the scenarios were predefined; the system did not adapt dynamically based on player behaviour or defensive strategies, leaving room for more interactive and adaptive simulations.

5. Zhang & Kumar, 2022 - Predicting Player Behaviour Using Supervised Learning
Zhang and Kumar (2022) applied decision tree and logistic regression models to predict player behaviour based on historical gameplay data, proving that supervised learning techniques can effectively capture behavioural patterns. Their approach improved game responsiveness but operated entirely offline, without integrating real-time prediction into active gameplay. This created a gap in live, adaptive enemy behaviour modelling.

6. Fernandes et al., 2022 - ML-Based Engagement Analytics in Educational Games
Fernandes et al. (2022) used machine learning to analyse and interpret player engagement levels in educational games, showing that ML analytics can help tailor experiences to individual learners. While effective for post-game analysis, the system did not perform real-time adaptation or influence gameplay while it was happening, limiting its applicability for immediate player feedback or dynamic challenge adjustment.

2.2 Functional Requirements

1) Core Gameplay Requirements
- The system must generate and display a network topology with multiple nodes and edges.
- The player must be able to place different types of defensive towers (Firewall, IDS, Honeypot) on allowed nodes.
- The system must allow removing or upgrading towers based on game rules.
- Enemy packets must spawn based on predefined or AI-selected enemy types.
- Enemy packets must navigate through dynamically generated or predefined paths.
- The system must detect collisions between towers and enemies and apply effects (damage, slow, diversion).
- The game must track packet outcomes (blocked, leaked, legitimate allowed).

2) AI & Adaptation Requirements
- The AI must record tower placement patterns during gameplay.
- The AI must analyze player strategy (e.g., Firewall heavy, IDS focused).
- The AI must spawn counter enemy types based on detected player strategy.
- The AI must switch network topology dynamically in Endless Mode.
- The system must provide post-game AI suggestions/tips based on performance metrics.

3) Level & Progression Requirements
- Each level must have specific configurations (enemy types, total packets, money, spawn rates).
- The game must display level introduction, objective, and tutorial text.
- The system must detect level completion based on success criteria (≥ threshold block rate).
- The system must detect level failure based on core-health reaching zero.
- Endless Mode must progressively increase difficulty and waves.

4) User Interface Requirements
- The UI must display HUD info including: money, score, core health, packets count, wave number, success rate.
- The UI must provide a tower selection menu with cost and description.
- The UI must visually highlight selected tower type.
- Popup notifications must be shown for wave changes, errors, and status updates.
- The system must visually display blinking nodes (start, goal, chokepoint guidance).

5) Pathfinding & Routing Requirements
- The system must compute all possible paths between source and goal nodes.

- The system must update paths dynamically if topology changes.
- The enemy must follow the chosen path smoothly along node coordinates.

6) Game State & Resource Requirements
- The system must track money, which decreases on tower purchase and increases on kills.
- The system must maintain a scoring system based on enemy outcomes.
- The system must support a limited number of active towers (tower cap).
- The system must track and update spawn timers dynamically.

7) Saving & Analytics Requirements
- The AI learning system must store gameplay session data.
- The system must log tower placements, enemy outcomes, and defense patterns.
- The system must load previous data to improve adaptive feedback.

2.3 Hardware Requirements

- Processor: Dual-Core CPU (Intel i3 / AMD equivalent)
- RAM: 4 GB
- Storage: 500 MB free space for project files, assets, and logs
- Graphics: Basic integrated GPU capable of rendering 2D graphics
- Display: 1366×768 resolution or higher
- Input Devices:
- Keyboard (for game controls)
- Mouse (for tower placement and interaction)

2.4 Software Requirements

- Operating System
  - Windows 10 or above
  - Linux (Ubuntu 20.04 or later)
  - macOS (for development only, if pygame supported)
- Programming Languages & Frameworks
  - Python 3.10+
  - Pygame for 2D rendering and game loop
  - NumPy for data handling and potential ML operations
  - JSON module for storing AI learning logs
  - Matplotlib for result visualization

- Development Tools
    - IDE / Code Editor:
    - VS Code
    - PyCharm
    - Sublime Text
- Version Control:
    - GitHub / Git
- Package Manager:
    - pip (Python package installer)
- Python Libraries Needed
    - pygame (main game engine)
    - random and math (built-in)
    - collections (for tracking AI learning stats)
    - typing (for structured type hints)

2.5 Summary

The literature highlights AI and gamified approaches in games and cybersecurity, including reinforcement learning for dynamic difficulty, hybrid AI frameworks, and ML-based player engagement analysis. Most studies are limited by offline learning, predefined scenarios, or lack of real-time adaptive behavior. To address these gaps, the proposed system combines real-time AI adaptation with interactive gameplay and dynamic network topologies. Functional requirements cover tower placement, enemy behavior, AI strategy analysis, level progression, and UI/UX features. Hardware needs are minimal (dual-core CPU, 4 GB RAM, basic GPU), and software relies on Python, Pygame, and supporting libraries. The system also logs gameplay data to improve adaptive AI feedback.

CHAPTER 3

DESIGN OF THE SYSTEM

The Network Defender system is designed as a gamified cybersecurity simulation that combines interactive tower-defense gameplay with AI-driven adaptive learning. It incorporates the following core design principles and methodologies:

System Architecture & Core Concepts

- Game Loop Architecture: The simulation operates on a continuous game loop that updates the game state, handles user input, computes AI decisions, and renders graphics in real-time.
- Event-Driven Interaction: All user actions, such as tower placement or upgrades, are handled as discrete events, ensuring responsive gameplay.
- Object-Oriented Entity Design: Key entities (Enemy, Tower, GameState, Player) are implemented as Python classes for modularity, maintainability, and scalability.
- AI-Assisted Difficulty Scaling: The AI monitors player strategy, adapts enemy types, and adjusts network topology dynamically to provide continuous challenge and learning opportunities.

3.1 Theory and Concepts

- Cyber Defense Fundamentals: The system integrates realistic cybersecurity elements such as firewalls, intrusion detection systems (IDS), and honeypots.
- Attack Evolution & Adversarial Learning: AI evolves attacks based on observed player strategies, simulating adaptive adversaries in a safe training environment.
- Real-Time Pathfinding & Waypoint Navigation: Enemy packets navigate through network topologies using dynamic pathfinding algorithms.
- Game Balancing & Resource Optimization: Budget constraints and level progression require players to strategize resource allocation effectively.

3.2 Dataset Description

The system generates a gameplay dataset during each session, which includes:

- Number and type of enemy packets (malicious or legitimate)
- Block success rate for defensive towers
- Player defense placement patterns and strategies
- Response time and resource utilization
- Timestamps of player actions.
- Wave number and enemy composition.
- Path chosen by AI enemies.
- Player tower placements and defense outcomes.
- Reward/penalty values assigned during reinforcement learning.
- Session-level summaries of success/failure rates.

This dataset allows post-game AI analysis to provide personalized recommendations for improving player performance.

3.3 Design and Methodology

1. Game Development: We develop the base tower defense game where packets (enemies) attack through network paths and the player defends using towers.
2. Reinforcement Learning Integration: A live feedback loop is implemented where the AI assigns rewards (+1) for successful breaches and penalties (−1) for failed ones, adjusting probabilities for future attacks accordingly.
3. Data Logging: We Capture every action, including defense placement, wave outcomes, and enemy behaviors, for analysis.
4. Supervised Learning Phase: Then we train offline models using logged data to predict which attack combinations most effectively challenge specific player patterns.
5. Dynamic Topology Switching: We Predefine multiple network layouts that change during gameplay to simulate evolving systems.
6. Feedback Visualization: End-of-session reports showing AI adaptation trends and suggesting optimal player strategies are displayed.

3.4 System Architecture

1. UI Layer: Manages HUD, menus, notifications, and visual cues for tower placement and network guidance.

2. Game Simulation Layer: Handles core game logic including packet movement, tower interactions, and resource tracking.

3. AI Learning Layer: Monitors gameplay, identifies patterns, adapts enemy strategies, and generates post-game recommendations.

4. Data Logging Layer: Records session data for analytics and AI feedback.

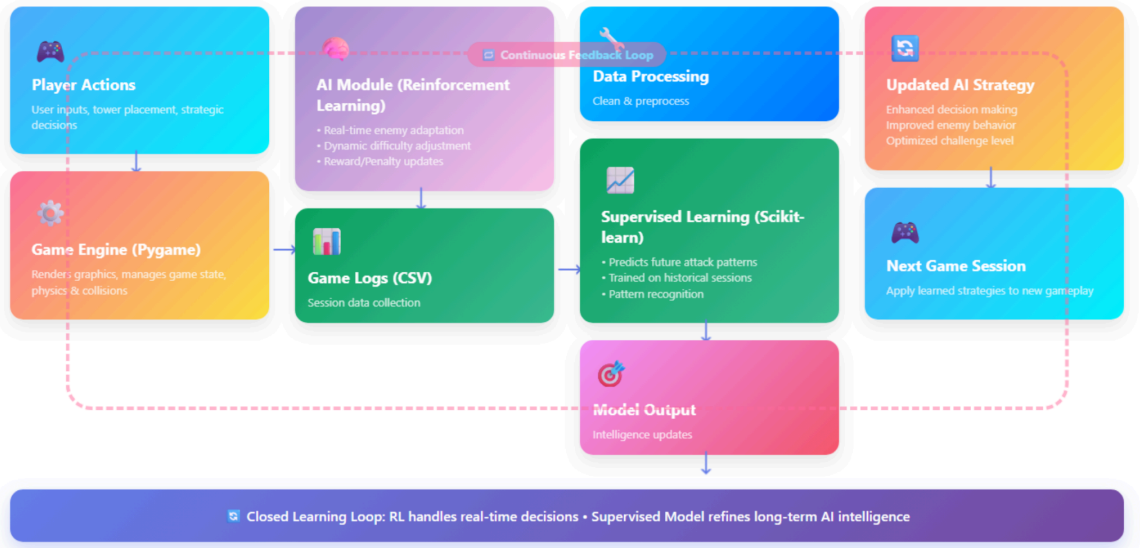

Figure 3.1 System Architecture

3.5 Tools and APIs

- Programming Language: Python

- Game Engine: Pygame
- Machine Learning Libraries: Scikit-learn (DecisionTreeClassifier, LogisticRegression)
- Data Handling: Pandas, NumPy
- Visualization: Matplotlib / Seaborn
- Storage: CSV-based local logs

3.6 Summary

Chapter 3 outlines the design and methodology of the Network Defender system, highlighting its gamified tower-defense gameplay integrated with AI-driven adaptive learning. It covers core concepts such as cyber defense mechanisms, real-time pathfinding, attack evolution, and resource optimization. The system captures detailed gameplay data for AI analysis, enabling dynamic enemy adaptation and post-game strategy recommendations. The architecture includes UI, game simulation, AI learning, and data logging layers, implemented using Python, Pygame, and machine learning libraries. This chapter emphasizes how design, methodology, and AI integration work together to create an interactive, adaptive cybersecurity training environment.

CHAPTER 4

IMPLEMENTATION AND TESTING

4.1 Implementation Requirements

- Python 3.8+
- Pygame library (pip install pygame)

4.2 Implementation Tool Features

- Real-Time Rendering Loop: The game updates and renders all entities, effects, and UI elements continuously.
- Modular Python Classes: Classes include Enemy, Tower, GameState, Player, and AI_Learning to ensure code reusability and clarity.
- AI-Integrated Enemy Decision Engine: Enemies are spawned based on AI analysis of player behavior to maintain challenge.
- UI-Based Interaction: Players interact via menus to place, upgrade, or remove towers, with visual feedback through blinking nodes and HUD notifications.
- Live Tower Menu: Shows tower types, cost, and descriptions.
- Packet Legend: Displays enemy packet types (malicious/legitimate).
- Guidance Nodes: Start, goal, and chokepoint nodes blink to guide player strategy.
- Dynamic Level Progression: Each level increases difficulty and challenges AI adaptation.

4.3 Code Snippets

- main.py – Handles the game loop, event management, and integration of all layers.

```python

def init_level(level_num):
    global game_state, tower_manager, enemies, game_flow_state, wave_popup_time
    wave_popup_time = None
    override_topo = None

    if LEVELS[level_num - 1].get("mode") == "ENDLESS":
        override_topo = generate_random_topology(
            node_count=8, edge_count=12,
            screen_width=SCREEN_WIDTH, screen_height=SCREEN_HEIGHT
        )

    game_state = GameState(level_num, override_topology=override_topo)
    game_state.tower_manager = TowerManager(tower_limit=10)  # tower cap 10
    tower_manager = game_state.tower_manager

    enemies.clear()
    game_flow_state = GAME_STATE_INTRO

def spawn_enemy():
    global enemies
    if game_state.level_number < 5:
        enemy_type = game_state.get_enemy_to_spawn()
    else:
        player_strategy = ai_system.analyze_defense_patterns()
        available_types = [
            t for t in game_state.level["enemy_types"] if t != "LEGITIMATE"
        ]
        legit_ratio = game_state.level.get("legitimate_ratio", 0)
        if legit_ratio > 0 and random.random() < legit_ratio:
            enemy_type = "LEGITIMATE"
        else:
            enemy_type = ai_system.select_counter_enemy_type(available_types, player_strategy)
    enemy = game_state.spawn_enemy(enemy_type)
    enemies.append(enemy)

def update_game():
    global game_flow_state, wave_popup_time
    if game_state is None or tower_manager is None:
        return

    for enemy in enemies:
        enemy.reset_slow()

    tower_manager.update(enemies)

    for enemy in enemies[:]:
```

```python
def draw_game():
    screen.fill((30, 30, 30))
    if game_state is None:
        font = pygame.font.SysFont(None, 48)
        screen.blit(
            font.render("Initializing...", True, (255, 255, 255)),
            (SCREEN_WIDTH // 2 - 100, SCREEN_HEIGHT // 2),
        )
        return
    ui.draw_network(
        game_state.nodes, game_state.edges, game_state.sources[0], game_state.goal_node
    )
    tower_manager.draw(screen)
    for e in enemies:
        e.draw(screen)
    ui.draw_hud(game_state)
    ui.draw_tower_menu(game_state.level["towers_available"], game_state.money)

    if game_flow_state == GAME_STATE_INTRO:
        ui.draw_level_intro(game_state)
    elif game_flow_state == GAME_STATE_COMPLETE:
        ui.draw_level_complete(game_state)
    elif game_flow_state == GAME_STATE_FAILED:
        ui.draw_level_failed(game_state)

def handle_tower_selection(mouse_pos):
    if game_state is None:
        return
    menu_x = SCREEN_WIDTH - 220
    menu_y = 120
    for i, tower_type in enumerate(game_state.level["towers_available"]):
        button_rect = pygame.Rect(menu_x, menu_y + i * 80, 190, 70)
        if button_rect.collidepoint(mouse_pos):
            ui.selected_tower_type = tower_type
def handle_tower_placement(mouse_pos, is_removal=False):
    if game_state is None or tower_manager is None:
        return
    node_id, node_pos = get_node_at_position(mouse_pos)
    if node_id is None:
        return

    # Enforce chokepoint placement restriction only on final level
    is_final_level = (game_state.level_number == len(LEVELS))
    if is_final_level and node_id not in game_state.chokepoints:
        print("X Tower placement only allowed on chokepoints in final level!")
        return

```

- game_state.py – Tracks core health, money, wave progression, and overall game state.

```python
import random
from enemies import ENEMY_CLASSES
from config import LEVELS


MAX_ACTIVE_ENEMIES = 8

class GameState:
    def __init__(self, level_number, override_topology=None):
        self.level = LEVELS[level_number - 1]
        self.level_number = level_number

        topo = override_topology if override_topology else self.level["topology"]

        self.nodes = topo["nodes"]
        self.edges = topo["edges"]
        self.sources = topo.get("sources", [topo.get("start", 0)])
        self.goal_node = topo["goal"]
        self.chokepoints = topo.get("chokepoints", [])

        self.money = self.level['starting_money']
        self.core_health = 100
        self.score = 0

        self.packets_spawned = 0
        self.packets_blocked = 0
        self.packets_leaked = 0
        self.legitimate_blocked = 0
        self.legitimate_allowed = 0

        self.spawn_timer = 0
        self.spawn_interval = self.level['spawn_interval']

        self.level_complete = False
        self.level_failed = False

        self.all_paths = None
    def set_topology(self, topology):
        self.nodes = topology.get('nodes', self.nodes)
        self.edges = topology.get('edges', self.edges)
        self.start_node = topology.get('start', self.start_node)
        self.goal_node = topology.get('goal', self.goal_node)
        self.chokepoints = topology.get('chokepoints', self.chokepoints)
        self.refresh_routing()  # Recalculate pathfinding if needed
        self.all_paths = None
        print("Topology updated.")

```

```python
    def update(self, enemies):
        print(f"Update called: packets_spawned={self.packets_spawned}, total_packets={self.level.get('total_packets')}, enemies={len(enemies)}")
        if self.core_health <= 0:
            self.level_failed = True
        if self.level.get('mode') != 'ENDLESS':
            total_packets = self.level['total_packets']
            if self.packets_spawned >= total_packets and len(enemies) == 0:
                success_rate = self.calculate_success_rate()
                if success_rate >= self.level['success_threshold']:
                    self.level_complete = True
                else:
                    self.level_failed = True
        wave_number = max(1, self.packets_spawned // 10)
        base_interval = self.level['spawn_interval']
        min_interval = 25
        self.spawn_interval = max(base_interval - wave_number * 3, min_interval)
        if self.packets_spawned != 0 and self.packets_spawned % 50 == 0:
            self.core_health = min(self.core_health + 15, 100)

    def should_spawn_enemy(self, current_enemies):
        if self.level.get('mode') != 'ENDLESS' and self.packets_spawned >= self.level['total_packets']:
            return False
        if len(current_enemies) >= MAX_ACTIVE_ENEMIES:
            return False
        self.spawn_timer += 1
        if self.spawn_timer >= self.spawn_interval:
            self.spawn_timer = 0
            return True
        return False

    def get_enemy_to_spawn(self):
        if self.level_number == 5 and 'LEGITIMATE' in self.level['enemy_types']:
            legit_ratio = self.level.get('legitimate_ratio', 0.3)
            if random.random() < legit_ratio:
                return 'LEGITIMATE'
        enemy_types = [t for t in self.level['enemy_types'] if t != 'LEGITIMATE']
        return random.choice(enemy_types)

    def enemy_destroyed(self, enemy):
        if hasattr(enemy, 'is_legitimate') and enemy.is_legitimate:
            self.legitimate_blocked += 1
            self.score -= 100
        else:
            self.packets_blocked += 1
            self.score += enemy.reward
            self.money += enemy.reward
```

- towers.py – Defines tower types, placement logic, upgrades, and effects on enemy packets.

```
class Tower:
    def update(self, enemies):
        if self.cooldown > 0:
            self.cooldown -= 1
        else:
            if self.effect == 'slow':  # IDS slows packets
                for enemy in enemies:
                    if getattr(enemy, 'is_legitimate', False):
                        continue
                    dist = math.sqrt((enemy.x - self.x) ** 2 + (enemy.y - self.y) ** 2)
                    if dist <= self.range:
                        enemy.apply_slow(0.3)  # stronger slow effect
                        self.cooldown = self.cooldown_time
                        break
            elif self.effect == 'attract':  # Honeypot attracts malicious packets
                for enemy in enemies:
                    if getattr(enemy, 'is_legitimate', False):
                        continue
                    dx = self.x - enemy.x
                    dy = self.y - enemy.y
                    dist = math.sqrt(dx * dx + dy * dy)
                    if dist < self.range * 2:
                        enemy.x += dx * 0.05
                        enemy.y += dy * 0.05

            elif self.damage > 0:
                for enemy in enemies:
                    if getattr(enemy, 'is_legitimate', False):
                        continue
                    dist = math.sqrt((enemy.x - self.x) ** 2 + (enemy.y - self.y) ** 2)
                    if dist <= self.range:
                        enemy.health -= self.damage
                        self.cooldown = self.cooldown_time
                        break

    def draw(self, screen):
        range_surface = pygame.Surface((self.range * 2, self.range * 2), pygame.SRCALPHA)
        pygame.draw.circle(range_surface, (*self.color, 30), (self.range, self.range), self.range)
        screen.blit(range_surface, (self.x - self.range, self.y - self.range))

        pygame.draw.circle(screen, self.color, (self.x, self.y), 15)
        pygame.draw.circle(screen, (255, 255, 255), (self.x, self.y), 15, 2)

        font = pygame.font.SysFont(None, 12)
        label = font.render(self.type[:3], True, (255, 255, 255))
        screen.blit(label, (self.x - 10, self.y - 5))
```

- ai_learning.py – Implements reinforcement and supervised learning to adapt enemy behavior based on player patterns.

```
    grid_xs_central = [
        border_margin_x + edge_node_margin + i * (usable_width // (grid_cols - 1))
        for i in range(grid_cols)
    ]
    grid_ys_central = [
        border_margin_y + edge_node_margin + j * (usable_height // (grid_rows - 1))
        for j in range(grid_rows)
    ]
    central_positions = [(x, y) for x in grid_xs_central for y in grid_ys_central]
    random.shuffle(central_positions)

    start_x = border_margin_x
    start_y = screen_height // 2
    goal_x = screen_width - right_ui_margin - border_margin_x
    goal_y = screen_height // 2

    positions = [(start_x, start_y), (goal_x, goal_y)]
    positions += central_positions[: node_count - 2]

    edges = set((i, i + 1) for i in range(node_count - 1) if not (i == 0 and i + 1 == 1))

    while len(edges) < edge_count:
        a, b = random.sample(range(node_count), 2)
        if a == b:
            continue
        edge = tuple(sorted((a, b)))
        if edge in edges or edge == (0, 1):
            continue
        edges.add(edge)

    goal_node = 1
    goal_edges = [e for e in edges if goal_node in e]
    goal_degree = len(goal_edges)
    attempts = 0
    while goal_degree < 2 and attempts < 20:
        candidate = random.choice([n for n in range(node_count) if n != goal_node])
        new_edge = tuple(sorted((goal_node, candidate)))
        if new_edge not in edges:
            edges.add(new_edge)
            goal_degree += 1
        attempts += 1

    edges = list(edges)

    sources = [0]
    for n in range(2, node_count):
```

- enemies.py – Manages enemy types, dynamic pathfinding, and interaction with towers.

```python
class Enemy:
    """Base enemy class."""

    def _init_(self, path: List[Tuple[int, int]], enemy_config: dict):
        self.path = path
        self.current_point = 0
        self.x, self.y = self.path[0] if path else (0, 0)

        # Stats from config
        self.health = enemy_config['health']
        self.max_health = enemy_config['health']
        self.base_speed = enemy_config['speed']
        self.speed = self.base_speed

        # State
        self.slow_factor = 1.0  # 1.0 = normal, 0.5 = 50% slow
        self.reached_goal = False
        self.is_legitimate = False

    def update(self):
        """Move along path."""
        if self.current_point >= len(self.path) - 1:
            self.reached_goal = True
            return

        target_x, target_y = self.path[self.current_point + 1]
        dx, dy = target_x - self.x, target_y - self.y
        dist = math.sqrt(dx ** 2 + dy ** 2)

        actual_speed = self.speed * self.slow_factor

        if dist < actual_speed:
            self.x, self.y = target_x, target_y
            self.current_point += 1
        else:
            self.x += actual_speed * dx / dist
            self.y += actual_speed * dy / dist

    def apply_slow(self, factor):
        """Apply slow effect from IDS."""
        self.slow_factor = min(self.slow_factor, factor)

    def reset_slow(self):
        """Reset slow effect each frame."""
        self.slow_factor = 1.0

```

Figure 4.1 Code Snippets

4.4 Testing Methodology

The system undergoes comprehensive testing to ensure performance, reliability, and effective learning:

- **Unit Testing:** Verifies individual components, such as tower effects, damage calculation, and pathfinding.
- **Integration Testing:** Ensures seamless interaction between game simulation, AI learning, UI, and data logging layers.
- **Performance Testing:** Assesses system stability under large numbers of packets and complex network topologies.
- **Game Balancing Testing:** Validates level difficulty, AI adaptation, resource constraints, and progression pacing.
- **User Feedback Testing:** Monitors player experience and learning retention, refining AI

recommendations accordingly.

4.5 Summary

The implementation of Network Defender integrates a modular Python-based game engine with AI-driven adaptive learning. Key features include live tower menus, packet legends, guidance nodes, dynamic level progression, and AI feedback for strategic gameplay. Major modules like main.py, game_state.py, towers.py, ai_learning.py, and enemies.py manage game logic, entity behavior, and AI adaptation. The system undergoes comprehensive testing, including unit tests for tower effects, integration tests for smooth layer interaction, performance tests under heavy loads, and game balancing tests to ensure progressive challenge and effective learning. Overall, the implementation and testing confirm a stable, interactive, and educational cybersecurity simulation environment.

CHAPTER 6

CONCLUSION

6.1 Conclusion

The Network Defender project successfully demonstrates that gamified cybersecurity simulations can significantly enhance user engagement, strategic thinking, and conceptual understanding of network defense mechanisms. By integrating real-time AI-assisted difficulty scaling, the system provides personalized learning experiences that adapt to individual player strategies, ensuring both replayability and continuous challenge. Players benefit from hands-on experience in deploying firewalls, IDS, and honeypots while responding to dynamically evolving cyber-attacks, which reinforces theoretical cybersecurity concepts in a practical, interactive environment.

6.2 Limitations

Despite these achievements, the system has certain limitations.

- No online multiplayer support, limiting collaborative or competitive gameplay.
- Lack of real packet capture or live network data integration, reducing realism in network traffic simulation.
- Limited AI-generated insights, providing basic feedback rather than advanced predictive analysis.
- Tower types and enemy packet variations are limited, reducing strategic complexity.
- No integration with real-world cybersecurity tools or monitoring systems.

6.3 Future Scope

- VR-based immersive simulations for highly realistic network defense training.
- Multiplayer modes enabling red-team vs blue-team exercises for collaborative and competitive learning.
- Configurable enterprise-level network models for advanced professional training.
- Expansion of AI capabilities to provide predictive analytics, strategy recommendations, and adaptive hints.
- Addition of more tower types, enemy packet variations, and complex attack patterns.
- Integration with real-time network monitoring tools for hybrid simulation and practical exposure.
- Cloud-based deployment for remote access and multi-user training sessions.
- Enhanced data visualization dashboards for live feedback on player performance and AI adaptation.
- Inclusion of scenario-based missions to simulate targeted attacks and real-world cybersecurity challenges.

References

1. Lee et al., “Applying Reinforcement Learning in Game AI for Dynamic Difficulty Adjustment,” *IEEE Access*, 2023.

2. Zhang & Kumar, “Supervised Learning for Player Behavior Prediction in Interactive Systems,” *Springer AI in Games Journal*, 2022.

3. Patel et al., “Gamified Cybersecurity Training with AI-Driven Scenarios,” *Computers & Security*, 2023.

4. Nguyen et al., “Hybrid AI Framework for Game Adaptation,” *Elsevier Expert Systems with Applications*, 2024.

5. Singh & Rao, “Dynamic Network Simulation Environments for Defense Training,” *ACM Simulation Modelling Practice and Theory*, 2023.

6. Fernandes et al., “Analyzing Player Engagement through Machine Learning in Educational Games,” *Journal of Learning Analytics*, 2022.