Autonomous AI for Boss Battles in Black Myth: Wukong

CS5446 Group 25 Project Report

Chen Huanhuan A0274951W E1124697

He Lurong A0268385M E1101684

Huang Weiqi A0290897H E1327928

Wang Qiang A0290887J E1327918

Abstract

Recent advances in action role-playing games (ARPGs) have introduced increasingly complex combat systems, posing unique challenges for artificial intelligence. This project presents an autonomous agent designed for boss battles within *Black Myth: Wukong* (BMW), focusing on real-time decision-making under partial observability.

We propose a hybrid architecture combining discrete state representation with deep reinforcement learning to address three key challenges: real-time state extraction, strategic decision-making, and efficient training. Our approach achieves a 90-95% success rate in simulated environments and demonstrates promising performance in real-game scenarios.

Through careful states design and accurate game state detection and classification, we show that our approach offers superior practical performance and training efficiency.

All implementation details and resources can be accessed at GitHub

Index Terms — Real-Time Decision Making, Partially Observable, Reinforcement Learning, Action Role-Playing Games (ARPGs)

1 Introduction

The rapid evolution of action role-playing games (ARPGs) has led to increasingly sophisticated combat systems that challenge both human players and artificial intelligence systems. *Black Myth: Wukong (BMW)*, with its remarkable success of 18 million copies sold within two weeks of launch, exemplifies this trend through its demanding combat system [1].

1.1 Game Context and Technical Challenges

In Black Myth: Wukong, boss encounters are central to gameplay, set in open, obstacle-free arenas that emphasize combat mechanics. Players begin at nearby checkpoints to save progress and recover. Upon defeat, they return to the checkpoint and navigate back to the boss arena, creating a repetitive yet adaptive gameplay loop.

In the arena, players engage the boss using the game's targeting system to maintain focus. Combat demands quick reflexes and tactical decisions, such as when to attack, dodge, or use resources like healing items. The interface provides critical information, including health bars, energy levels, and skill cooldowns, while boss animations signal incoming attacks, requiring players to anticipate and respond.

This structure fosters iterative learning, where players refine strategies through repeated attempts. The defined mechanics and fixed layout offer a robust framework for studying real-time combat challenges, including precise action timing, resource management, and adapting to dynamic enemy behavior.

1.2 Research Scope and Objectives

Our investigation focuses on the Youhun (Specter) boss encounter in BMW's early game. This encounter serves as an excellent research subject by combining basic attack patterns with challenging elements while minimizing confounding variables:

- Fixed arena layout enables focus on core combat mechanics
- Early-game positioning minimizes character progression variables
- Balanced mix of basic and advanced attack patterns

The project aims to develop an autonomous agent through three key objectives:

- **Real-time State Understanding:** Develop robust methods for processing complex visual input into actionable combat information.
- **Strategic Combat Mastery:** Create decision systems that balance immediate tactical responses with long-term resource management.
- **Efficient Training Framework:** Design practical approaches for learning complex combat strategies within real-time constraints.

2 Related Works

Our research builds upon key areas in game AI and reinforcement learning, addressing three critical challenges in BMW's combat system: real-time decision-making within strict 16ms constraints, partial observability due to dynamic visuals and occlusions, and scalable AI systems for complex combat.

2.1 Real-time State Understanding

In BMW's dynamic combat environment, accurately extracting critical information like boss attacks and health states in real time presents significant challenges. Baek and Kim [2] demonstrated the effectiveness of 3D ResNet for extracting spatiotemporal features in StarCraft II, which inspired our ResNet-based approach to boss attack recognition. Similarly, Lample and Chaplot [3] addressed partial observability in FPS games through deep reinforcement learning, informing our strategy for handling incomplete visual information during combat. We adapt these approaches to BMW's unique challenges by implementing a hybrid system that combines ResNet-based attack recognition with targeted state extraction for critical combat indicators.

2.2 Strategic Decision-Making

The complex nature of BMW's boss fights requires balancing immediate tactical actions with long-term resource management. Pope et al. [4] developed a reinforcement learning framework for air-to-air combat that uses reward shaping to integrate expert knowledge into decision-making. We extend this concept to BMW's combat system by designing a reward structure that encourages both optimal dodge timing and strategic resource usage. Our implementation particularly focuses on the critical balance between aggressive combat and survival, using a multi-layered reward system that accounts for both immediate combat success and long-term resource management.

2.3 Training Efficiency

Training AI for BMW's complex combat system presents unique challenges due to extended fight durations and precise timing requirements. Vinyals et al. [5] demonstrated with AlphaStar that well-designed simulation environments can enable efficient training while preserving essential mechanics. Building on this insight, we developed a specialized combat simulator that maintains BMW's core mechanics while accelerating the training process. Our simulation environment particularly focuses on replicating critical aspects of boss behavior patterns and combat timing, while abstracting less essential elements to enable rapid iteration and learning.

3 Framework Overview

3.1 POMDP Formulation

We model the combat system in BMW as a *Partially observable Markov Decision Process* (POMDP). While traditional approaches often rely on continuous state spaces and raw visual input, we demonstrate that a carefully designed discrete state representation can effectively capture essential combat dynamics while enabling stable learning.

3.1.1 State Space Design

The state space S, as shown in Table 1, is designed to capture key combat-critical information while maintaining tractability.

State Component	Representation	Range	Rationale
Boss State	Discrete	{0,1,2}	0: Non-attacking, 1: Attacking, 2: Frozen
Player Health	Discretized	[0,10]	Health percentage divided into 11 bins
Boss Health	Discretized	[0,5]	Health divided into 6 bins (20% each)
Medicine Count	Binary	{0,1}	Indicates healing item availability
Skill Cooldown	Binary	{0,1}	Freeze skill readiness
Combo Counter	Integer	[0,5]	Tracks successful consecutive attacks

Table 1: State space design capturing critical combat information.

Several traditional combat game states are intentionally omitted from our representation:

- **Positional Information:** The game's camera lock system ensures the boss remains centered, making absolute positions less critical. Relative positioning is implicitly handled through the dodge action system.
- **Animation States:** While full animation state tracking could provide additional information, the critical combat states (attacking/non-attacking/frozen) sufficiently capture decision-relevant information.
- **Boss Poise/Stagger:** Unlike games like Dark Souls, Black Myth: Wukong's boss poise system has minimal strategic impact in most encounters.

The discretization choices reflect a balance between state space complexity and strategic depth. For example, player health uses 10 bins because:

- Critical health thresholds occur at approximately 20%, 50%, and 80%.
- Healing items restore approximately 33% health.
- More granular bins showed no significant performance improvement in testing.

For the boss state, we combine both ResNet-based attack pattern classification and a 5-second vulnerability window inference after successful freeze skill activation. For the combo counter, we leverage the game's intrinsic combat mechanics where maintaining uninterrupted attack chains (3+ hits) enables enhanced damage output and temporary boss staggering effects. These strategic elements significantly influence the agent's combat decision-making.

3.1.2 Action Space

The action space A consists of five core combat actions:

$$A = \{\text{Wait, Attack, Dodge, Freeze, Recover}\}$$

The dodge action serves dual purpose for both evasion and positioning, while the camera lock system reduces the need for complex movement. These simplified actions are justified by combat analysis showing that high-level play primarily revolves around proper timing of basic attacks, dodges, and the freeze skill.

3.1.3 Observation Function

The observable elements from screen-based inputs directly map to our defined state space through targeted processing pipelines. However, several critical elements remain unobservable through our interface, including:

- Boss internal states and decision processes.
- Precise hitbox information and damage calculations.
- Internal cooldown timers and state machines (e.g., boss skill cooldown time).
- Audio cues and effects (e.g., a perfect dodge is indicated by sound effect).

Additionally, while certain elements like raw screen pixels and audio signals are technically observable, we deliberately exclude them due to processing complexity and limited strategic value in real-time decision making.

3.1.4 Transition Model

The combat system's transitions encompass both deterministic and stochastic elements. Core transition factors include:

- Health changes from combat interactions.
- Resource consumption and cooldown progressions.
- Combat state evolutions and phase transitions.

The stochastic nature of transitions manifests particularly in damage calculations and boss behavior patterns. For instance, identical attacks may result in different damage values based on hidden variables and timing precision. This uncertainty necessitates robust strategies that can adapt to varying outcomes rather than relying on fixed patterns.

These transition characteristics significantly impact our learning approach, requiring policies that balance consistent strategy execution with adaptive response to combat variations.

3.1.5 Reward System

Our reward system is designed to balance aggression and survival in combat. Placing too much emphasis on damage output leads to reckless behavior, while over-prioritizing survival results in passivity. To address this, we employ a layered reward system, combining terminal rewards for combat outcomes with intermediate rewards that provide consistent feedback during the fight.

Resource management is also a key aspect, with penalties for excessive medicine use to encourage efficient resource conservation, especially during critical moments.

A dynamic weighting system adjusts rewards based on the duration and context of the battle, helping the AI balance short-term actions with long-term goals. The discount factor $\gamma = 0.99$ ensures the AI considers both immediate and future rewards in its decision-making.

3.1.6 Formulation Analysis and Challenges

Our POMDP formulation addresses three fundamental challenges in real-time combat AI:

1. **State Space Complexity:** Continuous state spaces and complex transition dynamics in ARPG combat make direct learning approaches intractable. By using a discrete state representation focused on combat-essential information, we enable stable learning while preserving core strategic elements.
2. **Partial Observability:** The system operates solely on screen-based inputs, which contain numerous hidden states. A focused information processing pipeline, combined with temporal validation, ensures robust performance despite inherent state uncertainty.
3. **Real-time Constraints:** Dynamic combat scenarios demand rapid decision-making. A streamlined action space with well-defined execution windows balances tactical responsiveness with strategic coherence.

Through these targeted solutions, our formulation achieves a practical balance between computational feasibility and combat effectiveness, enabling reliable boss fight automation while maintaining learning stability.

3.2 System Architecture

Our system implements a modular pipeline architecture with three main processing stages and several supporting systems. The core components that warrant special attention include:

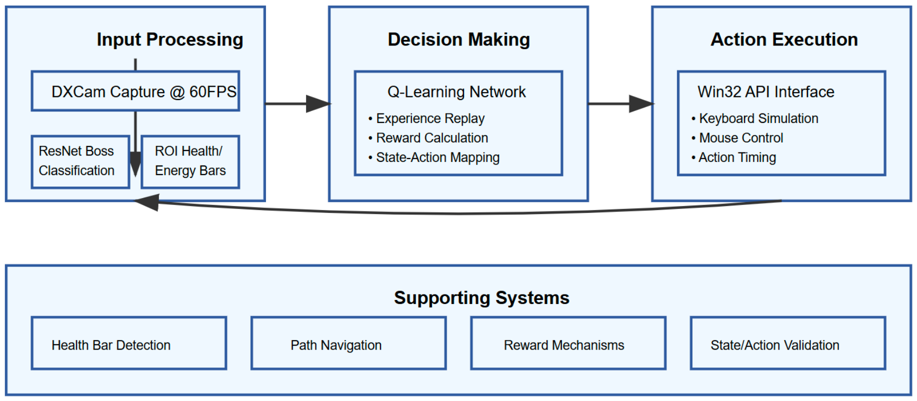


Figure 1: System Architecture

We developed a system featuring input processing with high-speed screen capture and precise action execution using the Win32 API, both of which are referenced from the *DQN_play_sekiro* framework [6]. Additionally, we implemented supporting systems for game state detection, boss behavior classification, automated navigation, and reward calculation.

The remaining components related to reinforcement learning and decision-making are discussed in detail in the subsequent section.

4 Methodology

Our methodology encompasses three components: automated navigation for consistent training sessions, state extraction from visual inputs, and training optimization. Each component addresses specific challenges while maintaining real-time performance requirements.

4.1 Automated Navigation System

We developed an automated navigation system that uses keystrokes to guide the player back to the boss after each defeat. Once the sequence of movement commands is executed, the system performs boss detection; if the boss is present, training resumes with the combat sequence. If the boss is not detected, the system employs teleportation to return to the starting point and retries the navigation process. This approach prioritizes consistent and reliable navigation over perfect precision, ensuring uninterrupted training sessions.

4.2 State Input Extraction

We do not have any APIs to directly read the states from memory, instead they are derived from game visuals and serve as the foundation for the AI's learning process. Their precision directly impacts the AI's ability to make timely and effective decisions in dynamic combat scenarios.

This section discusses two critical components of state input: Game State Detection and Boss Attack Prediction, both of which rely on visual processing to provide actionable data to the reinforcement learning framework.

4.2.1 State Detection for Key Combat Parameters

The state detection mechanism employs a sophisticated region-based extraction algorithm to capture key combat parameters, such as health, energy, magic, and skill values. This approach is primarily applied to character state information displayed on-screen in the game. By analyzing the continuity of grayscale values, the algorithm effectively extracts crucial metrics from the dynamic visuals, overcoming challenges like visual noise and fluctuating animations. Temporal smoothing ensures stability and accuracy, even during brief disruptions. Further details on the algorithm's implementation and challenges are provided in Appendix B and algorithm in Appendix D.2.

4.2.2 Boss Attack Prediction

We utilize a fine-tuned ResNet-18 model for combat state recognition, classifying boss attack states from visual input. ResNet-18 was chosen for its lightweight architecture, enabling real-time inference within 16ms, its residual connections for learning subtle animations, and pre-trained weights for effective feature extraction.

The model was trained on 30,000+ in-combat frames collected during gameplay. These frames were captured in real-time directly from the game, focusing on moments when changes in the player's health occurred. By analyzing these health changes, we inferred whether the boss performed an attack action, thereby labeling the frames with corresponding attack states. The frames were then manually verified to remove ambiguities and ensure high data quality. To further enhance the dataset, augmentation techniques were applied to simulate diverse combat conditions, such as motion blur and varying lighting. This targeted data collection and labeling process enabled the model to effectively learn the visual cues associated with boss attacks.

4.3 Model Training

4.3.1 Combat Simulation Environment

Due to the constraints of not being able to directly access precise game state data from API and the impracticality of lengthy training episodes (each lasting over 2 minutes with at least 1,000 episodes needed for a useful model), we developed a specialized combat simulation environment. This simulator abstracts the core mechanics of boss combat, focusing on essential decision-making elements critical for training the AI agent. By simplifying complex animations and non-critical interactions, we achieve faster training cycles, easier debugging, and a controlled environment to test various strategies and actions.

1) State and Action Space

The state and action space faithfully follows the previously formulated POMDP.

2) Transition Dynamics Model

Our transition model employs a hybrid deterministic-stochastic approach that captures combat uncertainty while maintaining learning feasibility. The transition function $T(s'|s,a)$ incorporates several game-specific mechanics. Detailed design is documented in Appendix C.

3) Simulation Effectiveness Analysis

To evaluate the effectiveness of our simulation environment and the trained AI agent, we conducted a series of test cases that assessed the agent's decision-making in various combat scenarios.

The agent's responses in the test cases demonstrate that it has effectively learned key combat strategies: prioritizing survival by dodging or healing when health is low and the boss is attacking; maximizing damage by attacking when the boss is frozen or not attacking to leverage higher success rates and combo multipliers; managing resources efficiently by judicious use of the freeze skill and healing items, avoiding unnecessary or wasteful usage; and adapting to cooldowns and resource limitations by refraining from attempting actions that are unavailable, indicating an awareness of game mechanics and strategic decision-making.

4) Error Simulation

To simulate real-world uncertainties and detection errors, we intentionally introduced randomness in the boss's ability to cause damage even when not in an attacking state. This reflects potential inaccuracies in state detection due to the limitations of image recognition tools (e.g., ResNet models) used to interpret game visuals.

- **Error Simulation in Boss Attacks:** The 20% chance of the boss causing damage when in a non-attacking state encourages the agent to develop strategies that are robust to state detection errors, improving its real-world applicability.
- **Stochastic Elements:** By incorporating probabilistic outcomes in attacks, dodges, and state transitions, the simulation ensures that the AI cannot rely on deterministic patterns and must adapt to a variety of scenarios. This randomness compels the agent to develop flexible strategies that can handle unexpected events.

4.3.2 Training Process

We utilized the simulator for initial training and debugging, after which the trained models were deployed to the real environment for fine-tuning.

1) Key Elements:

- **State History Integration:** While our state space is discrete, the policy π_ϵ considers a short history buffer h_t to capture temporal patterns in boss behavior.
- **Priority-Adjusted Experience Replay:** Transition priority p_t incorporates both TD-error and combo state, ensuring the agent learns from both rare high-reward sequences and critical combo opportunities.

2) Hyperparameter Selection: Key parameters were selected to balance learning stability with combat performance:

- **Learning Rate** ($\alpha = 5e$-4): Balances stability with convergence speed
- **Discount Factor** ($\gamma = 0.99$): Matches combat duration and reward delay
- **Replay Buffer Size** ($5e4$): Maintains diverse experience sampling

For exploration-exploitation balance, we implemented a dynamic epsilon decay:

$$\epsilon_{\text{decay}} = \frac{\epsilon - \epsilon_{\min}}{num\_episodes}$$

where $\epsilon_0 = 1.0$, $\epsilon_{\min} = 0.01$. This approach ensures aggressive initial exploration while allowing faster convergence when the agent demonstrates consistent combat success.

The implementation achieves stable learning across different boss patterns while maintaining real-time decision making within our 16ms target window, demonstrating combat behaviors that align with expert gameplay principles despite the simplified state representation.

3) Real World Training Setup Our real world training environment prioritizes consistency and reproducibility. The game is configured at a fixed 1280x720 resolution in windowed mode to ensure stable visual input processing.

4.3.3 Alternative Methods Tried

We refined our approach through several key iterations. Before separating the ResNet model and introducing the combat simulator with predefined states, we experimented with alternative methods. This section describes the limitations and preliminary results of these approaches. Notably, both methods involved feeding raw screen images directly into the network; we refer to these as end-to-end models.

1) Approach 1: DQN with Convolutional Architecture

In our first implementation, we directly used raw visual frames as input without any pre-processing. This approach had two main benefits: it was simple to implement and it preserved all the information, avoiding the loss of important details from manual feature selection.

However, this approach encountered some key issues. The high-dimensional input space (128x128×3) required extensive training data, and CNN processing introduced latency that exceeded our 16ms decision window target. Additionally, gradient updates were unstable due to the sparse nature of combat rewards.

2) Approach 2: DDQN with ResNet

To address the limitations of our initial approach, we implemented Double DQN (DDQN) with ResNet feature extraction, introducing several key improvements. DDQN's decoupled action selection and evaluation networks proved valuable in handling sparse, delayed reward signals, and reducing errors in state-action value estimation during long episodes. This approach also addressed the high variance in combat outcomes.

Additionally, ResNet's skip connections improved the feature extraction process by preserving low-level visual details, such as attack animations, while efficiently handling the spatial complexities in combat scenes.

However, this iteration still faced some challenges. Training time exceeded practical limits, taking more than 30 days to converge. Real-time inference continued to fall short of performance targets, and feature extraction remained sensitive to visual noise.

5 Evaluation

5.1 Model Evolution and Performance Results

In this section, we first present the results of our preliminary experiments using one of the two alternative approaches, which highlight the challenges we faced when training such end-to-end models. Then, we showcase the results of our final model that utilizes discrete states, ResNet for classification and a simulated environment, demonstrating significant training progress and performance improvement.

5.1.1 DDQN+ResNet Performance

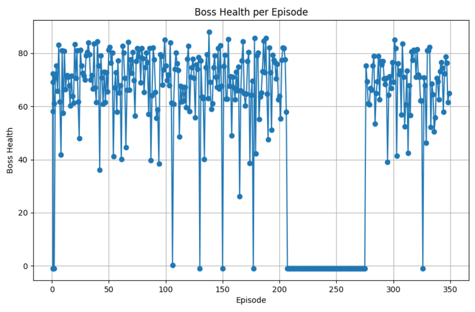


Figure 2: Boss Health Per Episode for DDQN+Resnet (End-to-End) In Real Combat

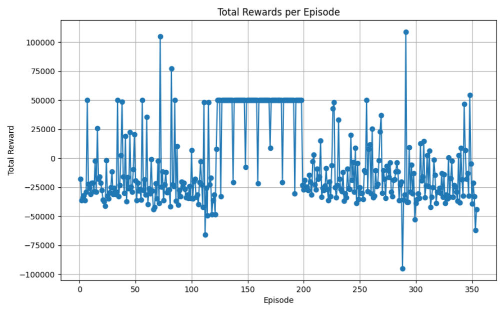


Figure 3: Total Reward Per Episode for DDQN+Resnet (End-to-End) In Real Combat

The DDQN+ResNet implementation with demonstrated several challenges in achieving stable performance. Figures 4 and 5 illustrate the performance metrics over approximately 350 episodes:

- Boss health fluctuated significantly between 40–80% with no clear convergence trend.

- Total rewards showed high volatility, ranging from -75,000 to +100,000.
- A notable anomaly appears between episodes 200–270, showing a flat line at 0% boss health, this is actually due to the auto-navigation system failed to navigate to the boss. Highlighting one of the challenges in training.

The limited episode count (approximately 350) due to extended training durations per episode significantly constrained the model's learning potential. This practical limitation, combined with the high variance in performance, motivated our transition to a discrete state space approach with simulator training.

5.1.2 Discrete State Model Performance

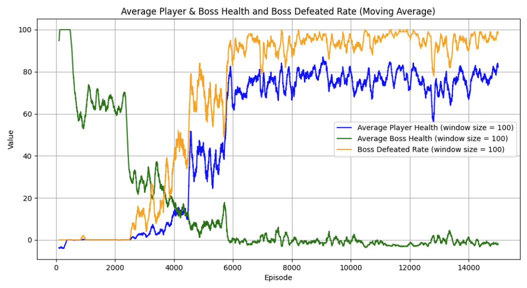


Figure 4: Game Statistics Per Episode for DQN with Discrete State In Simulation

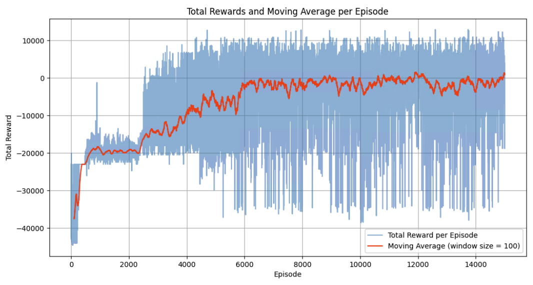


Figure 5: Total Reward Per Episode for DQN with Discrete State In Simulation

The transition to discrete state representation with simulator training showed markedly improved learning characteristics, as evidenced in Figures 4 and 5. The learning progression can be divided into distinct phases:

Phase	**Episodes**	**Key Characteristics**	**Performance Metrics**
Initial Adaptation	0–2000	Sharp improvement from negative rewards	Learning basic survival
Rapid Learning	2000–4000	Consistent upward trend	Developing combat strategies
Optimization	4000–6000	Increased stability	Refining tactics
Convergence	6000+	Maintained positive performance	Stable 95% win rate

Table 2: Learning progression of discrete state model.

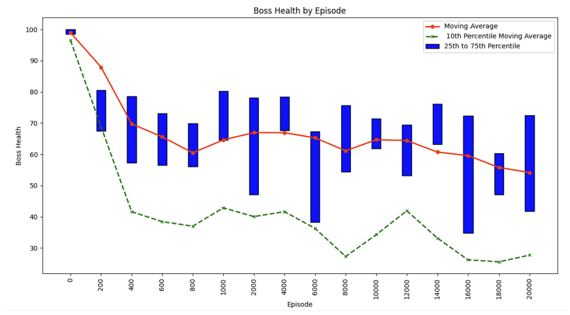
Additionally, training in the actual game environment was highly time-consuming; completing 2,000 episodes could take up to four days due to lengthy episodes and slow data collection. With our combat simulator, we can now run 20,000 episodes in just 30 minutes. This significant reduction in training time accelerates the debugging process and allows for rapid optimization of the model's performance. For example, we realized the importance of incorporating a replay buffer in the Deep Q-Network algorithm only after conducting several 20,000-episode training sessions and comparing the results.

The performance metrics from the simulator showed impressive final results:

- Player health stabilized around 70–80%, indicating effective survival strategies and health management by the agent.
- Boss defeated rate maintained at approximately 90–95%, demonstrating proficiency in combat and the ability to successfully execute offensive tactics.

5.2 Real-World Performance Analysis

The transition from simulator to actual game environment revealed important insights about model robustness and generalization. Figures 6 and 7 present crucial evaluation data from multiple real game fights at different training stages.



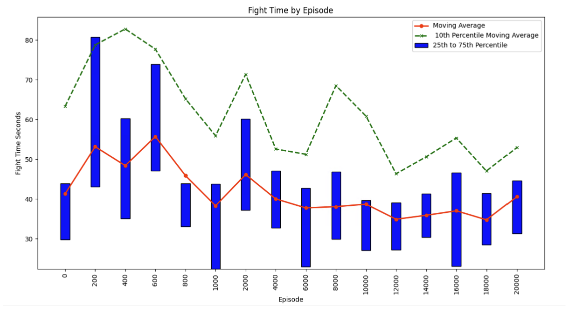


Figure 6: Boss Health Per Episode In Real Combat

Figure 7: Fight Time Per Episode In Real Combat

5.2.1 Boss Health Progression

Figure 6 illustrates a significant performance improvement during the first 800 episodes. Although there are consistent enhancements up to 20,000 episodes, we believe that the accuracy of state readings and classifications becomes the primary limiting factor affecting further performance gains.

5.2.2 Fight Duration Analysis

Figure 7 reveals an interesting pattern in combat duration:

- Initial increase (0–200 episodes): Combat duration extended from 40s to 55s.
- Peak duration (200–600 episodes): Maintained longer engagement times.
- Progressive decrease (600+ episodes): Duration optimized to 35–40s.

This pattern suggests a sophisticated evolution of combat strategy, from initial survival-focused behavior to more efficient and aggressive tactical approaches, see Table 3

Phase	**Fight Duration**	**Boss Health**	**Strategic Characteristics**
Defensive Learning	50–55s	70–80%	Focus on survival mechanics, basic pattern recognition
Tactical Development	40–45s	60–65%	Balanced aggression, improved damage opportunities
Strategic Optimization	35–40s	55–60%	Optimized damage dealing, aggressive yet calculated approach

Table 3: Strategic evolution during training.

5.3 Performance Gap Analysis

The disparity between simulator and real-game performance can be attributed to two primary factors:

5.3.1 State Reading Accuracies

The table below presents our estimated accuracies for various state readings. Although these accuracies are relatively high, they are still insufficient for optimal performance. For example, a 90% accuracy rate in classifying boss attacks means that 100 out of every 1,000 frames may be misclassified. This is particularly problematic because the most challenging aspect of this boss encounter is that failing to dodge when the boss is casting its ultimate attack can lead to the player being defeated within seconds.

Component	**Estimated Accuracy**	**Impact on Performance**
State Bar Detection	99%	State assessment errors
Boss Attack Classification	85–90%	Tactical response delays

Table 4: State Reading Accuracies

5.3.2 Environmental Complexity Comparison

Factor	Simulator	Real Game
State Information	Perfect accuracy	Processed from visual input
Timing	Consistent	Variable processing delays and animations
Boss Behavior	Simplified patterns	Complex animation transitions
Execution Delay	0	Different actions take different time

Table 5: Environmental complexity comparison.

Furthermore, Table 5 presents additional factors that further impact performance.

6 Conclusion

In this work, we developed:

- A practical framework for creating autonomous agents in commercial action role-playing games (ARPGs) that balances theoretical rigor with practical implementation feasibility.
- Innovative real-time state extraction methods that achieve robust performance under strict timing constraints.
- A combat simulation architecture that enables efficient training while preserving essential strategic elements of gameplay.
- Empirical evidence demonstrating the effectiveness of discrete state representations in complex, real-time combat scenarios.

Our results demonstrate that carefully designed discrete state spaces can effectively capture complex combat dynamics while enabling practical training and deployment.

6.1 Limitations

Several key limitations warrant consideration:

- State extraction reliability remains below 95%, affecting real-time decision quality
- Limited ability to generalize to significantly different boss patterns
- Performance degradation when encountering edge cases not covered in training
- Dependency on specific visual indicators constrains cross-game applicability

6.2 Future Directions

Building upon our current work, we identify several key areas for future exploration:

- **Generalization to Multiple Bosses**: Expand the learning process and model to handle multiple bosses with varying behaviors and attack patterns, enhancing the agent's adaptability and robustness across diverse combat scenarios.
- **Enhanced Path Navigation Using LLMs**: Integrate Large Language Models (LLMs) to develop a more stable and intelligent path navigation system, allowing the agent to interpret in-game cues and adjust its route dynamically for improved efficiency.
- **Advanced Boss Action Detection Techniques**: Employ methods like OpenPose for more accurate boss action detection, increasing state extraction reliability and enabling the agent to make better-informed decisions during combat.

These future directions aim to further improve the agent's performance and applicability in complex real-time gaming environments.

References

[1] S. Shah. Black myth wukong sells 18 million copies, becoming one of the best-selling games ever. *Evening Standard*, September 5 2024. Accessed online.

[2] Insung Baek and Seoung Bum Kim. 3-dimensional convolutional neural networks for predicting starcraft results and extracting key game situations. *Plos one*, 17(3):e0264550, 2022.

[3] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[4] Adrian P Pope, Jaime S Ide, Daria Mićović, Henry Diaz, David Rosenbluth, Lee Ritholtz, Jason C Twedt, Thayne T Walker, Kevin Alcedo, and Daniel Javorsek. Hierarchical reinforcement learning for air-to-air combat. In *2021 international conference on unmanned aircraft systems (ICUAS)*, pages 275–284. IEEE, 2021.

[5] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

[6] Analoganddigital (GitHub username). Dqn_play_sekiro. `https://github.com/analoganddigital/DQN_play_sekiro`, 2021. GitHub repository.

Appendix

A Source Code

Our source code and some of the trained models can be found at: https://github.com/allenwq/cs5446-wukong-ai

B State Detection Algorithms

B.1 Introduction

This detection algorithm is designed to identify player and boss status bars, such as health, energy, magic and skill bars, which are crucial for calculating rewards and making decisions based on combat status. While this method is applicable to various on-screen parameters, we use the health bar as an example to demonstrate the approach, which includes region-based extraction, grayscale filtering, and continuity analysis.

B.2 Implementation

B.2.1 State Bar Characteristics

The algorithm design is based on the distinct characteristics of state bars in the game. For instance, a health bar is a thin, rectangular strip with clear visual cues. When full, it appears white, while the unfilled portion is black, with a clear boundary separating the two. The player's health bar is always positioned at the bottom-left of the screen, while the boss's health bar is located directly above it, centered horizontally in the middle of the screen. Various animations, such as green regions for healing, red regions for damage, and red flashing at critical health, add complexity by dynamically altering the bar's appearance.

B.2.2 Detection Logic

The health bar's fixed position in combat mode allows the use of region-based extraction to isolate it from the game screen. The resolution is predefined (1280 * 720), and the health bar's coordinates are statically mapped relative to the top-left corner. Variations in health bar length (e.g., different bosses) are handled by manually adjusting the coordinates as parameters.

The algorithm performs the following steps:

1. **Preprocessing:** The algorithm converts the input image to grayscale to simplify analysis and reduce color-based noise. It crops 2 pixels from each side to eliminate boundary interference, extracts the middle 7 rows around the health bar's center line to focus on its most stable region, and filters pixels to a grayscale range of 90-255 to isolate relevant features.
2. **Continuity Analysis:** Continuity metrics are used to measure row and column smoothness within the health bar. Both the filled and unfilled sections are expected to have uniform grayscale values under normal conditions, regardless of animations. By combining row and column continuity, the algorithm identifies smooth, consistent regions indicative of health bars.
3. **Health Bar Existence Check:** The algorithm confirms the presence of a health bar if row continuity exceeds 0.3 and the proportion of valid grayscale pixels is above 0.1. Otherwise, the bar is considered undetected.
4. **Health Percentage Calculation:** If the bar is detected, the health percentage is calculated as the ratio of valid pixels to the total pixels in the region. This percentage provides a reliable measure of the player's or boss's current health, forming the basis for gameplay decisions.

B.3 Challenges

B.3.1 Challenge 1: Detecting the Health Bar and Calculating Its Percentage

Detecting health bars is complicated by environmental noise and dynamic animations. The game environment introduces distractions like trees, grass, and terrain textures, while the bar's semi-transparent gray patterns can blend into the background during player movement. Animations such as green effects for healing, red effects for damage, and red flashing at critical health further obscure boundaries, making edge-based detection methods unreliable. The algorithm addresses these challenges using continuity metrics, focusing on smoothness within the bar rather than its edges, making it robust against noise and animations.

B.3.2 Challenge 2: Ensuring Stability in Edge Cases

Rapid environmental changes or dynamic visual effects can occasionally cause the algorithm to miss detecting the health bar. This results in brief interruptions in detection, such as a correct series of health percentages being momentarily disrupted. To address this, the algorithm maintains a history of the last 3 frames and uses the mode (most frequent value) to filter out anomalies. By smoothing these interruptions, the algorithm ensures consistent and reliable output, even in dynamic scenarios.

C Combat Simulation Environment State Transitions

1. **Player Actions and Outcomes**
   - Regular Attack: The success probability of an attack (P_{hit}) depends on the boss's state:
     - When the boss is frozen (boss_state $= 2$): $P_{\text{hit}} = 1.0$
     - When the boss is non-attacking (boss_state $= 0$): $P_{\text{hit}} = 0.4$
     - When the boss is attacking (boss_state $= 1$): $P_{\text{hit}} = 0.05$

     Upon a successful hit, the boss' health decreases by damage calculated with a combo multiplier:

     $$\text{Damage} = \text{Base Damage} \times (1 + \text{Combo Multiplier} \times \text{Consecutive Attack Count})$$

     where the base damage is a predefined random value.
   - Dodge: The player has a 70% chance to successfully evade an attack: $P_{\text{dodge}} = 0.7$
   - Use Freeze Skill: Can only be used if not on cooldown (freeze_skill_cd $= 0$). If the boss is attacking (boss_state $= 1$), the boss becomes frozen (boss_state $= 2$) for 5 seconds. The skill then enters a cooldown period of 30 seconds.
   - Consume Medicine: If the player has medicine left and is not in full health, taking the medicine restores 33% health.
2. **Boss Actions and State Transitions**
   - Boss Attack: The boss can cause damage to the player with probabilities:
     - When the boss is attacking: $P_{\text{boss\_hit}} = 0.9$
     - When the boss is non-attacking: $P_{\text{boss\_hit}} = 0.2$

     The 20% chance of the boss causing damage when not attacking simulates potential errors in state detection due to limitations in image recognition models used in the actual game.
   - State Changes: The boss randomly transitions between states when not frozen:

     $$P(\text{boss\_state} = 1 \mid \text{boss\_state} = 0) = 0.5$$
3. **Timers and Cooldowns**
   - Freeze Duration: When the boss is frozen, it remains in that state for 5 seconds.
   - Freeze Skill Cooldown: After use, the freeze skill is unavailable for 30 seconds.

D Specific Algorithms

D.1 Reinforcement Learning Training Loop

Algorithm 1 Reinforcement Learning Training Loop

```
Initialize environment $\mathcal{E}$, context $\mathcal{C}$, agent policy $\pi_\theta$, and replay buffer $\mathcal{D} \leftarrow \emptyset$
Load pre-trained classifier $f_\phi$ **for** *episode* $e = 1$ *to* $E$ **do**
        Initialize state $s_0 \sim \mathcal{E}$, set total reward $R_e \leftarrow 0$ **while** *not terminal,* $s_t \notin \mathcal{S}_{terminal}$ **do**
        Extract raw observation $\mathcal{O}_t$ from environment
Classify boss state: $\text{BossState}_t \leftarrow f_\phi(\mathcal{O}_t)$
Augment state: $s_t \leftarrow \text{discretize}(\mathcal{C}, \text{BossState}_t)$
Select action $a_t \sim \pi_\theta(a|s_t)$                                        ▷ Policy-based action sampling
Execute $a_t$ in $\mathcal{E}$, observe next observation $\mathcal{O}_{t+1}$ and reward $r_t$
Classify next boss state: $\text{BossState}_{t+1} \leftarrow f_\phi(\mathcal{O}_{t+1})$
Update context $\mathcal{C} \leftarrow \mathcal{C}(a_t)$
Augment next state: $s_{t+1} \leftarrow \text{discretize}(\mathcal{C}, \text{BossState}_{t+1})$
Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$
Accumulate reward $R_e \leftarrow R_e + r_t$ **if** $|\mathcal{D}| \geq N_{batch}$ *and* $t \mod N_{update} = 0$ **then**
        Sample mini-batch $\mathcal{B} \sim \mathcal{D}$
Compute target $y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)$
Perform gradient update: $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\pi_\theta, \mathcal{B})$        ▷ Q-learning or policy gradient update

$s_t \leftarrow s_{t+1}$ **if** *death detected, health* $\leq 0$*, or BossState*$_t$ $=$ *critical* **then**
        Set $s_t \in \mathcal{S}_{\text{terminal}}$                                        ▷ End condition met


Evaluate total reward $R_e$ **if** *episode % N*$_{target\_update}$ $== 0$ **then**
        Update target network weights: $\theta^- \leftarrow \theta$                ▷ Target network synchronization
```

D.2 State Bar Detection

Algorithm 2 State Bar Detection

Input: Frame $frame$, ROI (sx, sy, ex, ey)
Output: Health percentage $health\_percent$
Extract ROI: Crop $frame$ to (sx, sy, ex, ey).
Preprocess:

- Convert ROI to grayscale.
- Extract middle rows and filter grayscale pixels in $[90, 255]$.

Continuity Check: Compute row and column smoothness.

- If $continuity > 0.3$ and $current\_health\_percentage > 0.1$, proceed; else $health\_percent = -1$.

Health Calculation: $health\_percent = \frac{valid\_pixels}{total\_pixels} \times 100$.
Smooth Results: Update history buffer and filter with mode of recent frames.
Return $health\_percent$.

E Team Contribution

Table 6: Workload Breakdown

Team Member	**Main Responsibilities**
Chen Huanhuan	Model study, evaluation/testing and demo/report
He Lurong	State detection algorithm design and code/demo/report
Huang Weiqi	Image recognition design and code/-Training data collection and processing
Wang Qiang	Key models and simulator development, result benchmark, demo/report

***Disclaimer: Use of AI Assistance**

This report was developed with the support of ChatGPT, an AI language model. The following contributions were made with the assistance of ChatGPT:

- **Language Refinement:** Editing and refining the text for clarity, coherence, and grammar.
- **Structural Suggestions:** Proposing outlines and improving organization where needed.