



Real-Time Log-Based Novice Recognition and Dynamic Support System

**Jinyoung Maeng¹; Junhui Kim²; Dowon Seo³;
Jeongmin Hwang⁴; Seungjae Lee^{5*}**

^{1,2,3,4,5} Computer Engineering Department, Sunmoon University, Korea

¹ mjyoung1226@sunmoon.ac.kr; ² junhui0119@sunmoon.ac.kr; ³ dowon1942@sunmoon.ac.kr;

⁴ gwa2021243113@sunmoon.ac.kr; ^{5*} (corresponding author) leeko@sunmoon.ac.kr

DOI: <https://doi.org/10.47760/ijcsmc.2025.v14i09.007>

Abstract: This study proposes Continue AI, an AI support system for novice gamers. To address the problem of novice player attrition in the Vampire Survivors-like genre, we developed a real-time personalized support system that integrates skill classification based on RandomForestClassifier, danger detection with the Danger_score algorithm, and quest generation using Ollama LLM. By implementing efficient real-time communication through a Unity-Python-Redis three-tier architecture and Protocol Buffers, we aim to improve game accessibility for beginners.

Keywords: Game AI, Asynchronous TCP Server, Protocol Buffers, Real-time Data Processing, User Behavior Prediction, Personalized Support System

I. INTRODUCTION

Achieving an appropriate difficulty balance in games is crucial for player retention. Previous research indicates that game difficulty directly impacts player enjoyment, with higher satisfaction observed when the difficulty aligns with players' perceived experience levels rather than their actual abilities [10]. Therefore, it's vital to prevent players from feeling frustrated and abandoning games due to excessive difficulty [1]. Existing static difficulty adjustment methods fail to account for individual player skill differences, offering no fundamental solution.

This study proposes Continue AI, a real-time personalized support system utilizing AI technology. Its core features include a 7-level skill classification based on RandomForestClassifier, risk situation detection using the Danger_score algorithm, and dynamic quest generation via Ollama LLM. The system is built on a three-tier architecture comprising a Unity client, a Python server, and a Redis database. Efficient data serialization through Protocol Buffers and real-time log collection at 1-second intervals enable immediate AI analysis. Our goal is to design a system that allows even novice players to enjoy games more stably and immersively.

II. RELATED WORKS

A. Dynamic Difficulty Adjustment Systems

Traditional single-player games often require players to select a static difficulty (e.g., Easy, Normal, Hard) at the start, which can lead to a mismatch between an individual player's actual skill and the game's challenge level. To address this issue, Dynamic Difficulty Adjustment (DDA) techniques have been proposed [2]. Hunicke et al. presented the Hamlet system, built on the Half-Life engine, which statistically analyzes players' real-time resource consumption (health, ammunition, etc.) and dynamically adjusts difficulty, such as enemy attack power or resource placement, when an anticipated failure situation (shortfall) is detected [2].

They implemented a policy-based structure to control the frequency and intensity of adjustments, aiming to keep players in a state of flow. Meanwhile, Kennerly introduced a data mining approach for MMOG environments, analyzing play logs offline to balance in-game economies, detect cheaters, and improve customer retention [3]. There have also been attempts to dynamically adjust enemy abilities based on real-time data like hit rates or play time using fuzzy logic [9]. However, these existing studies, based on predefined rules or specific theories, have limitations in comprehensively judging player skill in complex game situations.

B. Real-time Player Support Systems

Recently, the focus has shifted from simple difficulty adjustment to systems that analyze player states and behaviors to provide personalized support. For example, systems like FlexBot offer enemy behavior changes tailored to the player's style [4], and LLM-based games dynamically generate personalized narratives through text-based input, presenting a new paradigm for interactive fiction games [5].

This research trend aligns with the direction of ContinueAI. ContinueAI goes beyond mere palliative interventions to prevent failure; it is designed to identify novice players based on real-time logs and statistical data, and then provide appropriate support buffs or natural language quests to maintain game immersion and a sense of accomplishment for those users.

III. SYSTEM ARCHITECTURE

A. System Architecture

The Continue AI system proposed in this study adopts a three-tier architecture consisting of a Unity client, an asynchronous server implemented in Python, and a Redis database. (Fig. 1) illustrates the overall structure of the Continue AI system.

Game-play data generated by the Unity client is serialized using Protocol Buffers and then transmitted to the server via TCP communication. The server processes the received data, stores it in the Redis database, and provides it as analysis data to the AI module.

The AI module analyzes the accumulated data in Redis to understand player patterns and determines appropriate support measures. The determined support information is sent to the client via the server, and the client provides appropriate assistance and displays it on the UI based on the received support information.

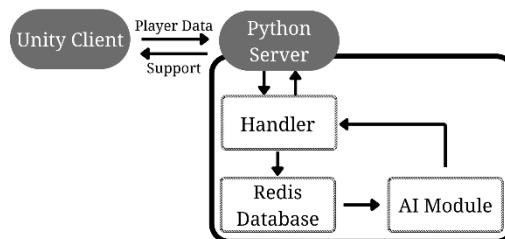


Fig. 1 ContinueAI System Architecture

Through this series of processes, the Continue AI system can analyze player game data in real-time and provide appropriate support based on the results.

B. Simulation Environment for Data Collection and Application

This simulation environment includes multiple interactive game systems to collect complex player behavior data. A wave system that periodically spawns monsters provides continuous challenges to the player, with elite and boss monsters posing significant threats.

The time-based difficulty adjustment feature scales up monster abilities in proportion to elapsed game time, recording the player's adaptation to changing threats.

Player growth and selection processes are also datified. Items (equipment, accessories) can only be acquired through leveling up, loot boxes, or shop NPCs, and all skills, except basic attacks, are bound to equipment and activate automatically.

C. Game Player Data Information

The proposed system has a structure that collects and analyzes real-time user data generated during gameplay. The Unity-based client extracts player behavior and status data every 1 second, and this information is serialized using Protocol Buffers.

The serialized data is transmitted to the Python asynchronous server via TCP communication, and the server processes the received information before storing it in the Redis database.

Redis stores data chronologically, with each player's UUID as the key. This allows the system to quickly query player status changes and reliably secure the data needed for AI analysis.

D. AI-based Real-time Buff Recommendation and Application System

In this study, we implemented an AI support system that analyzes real-time player log data to recognize dangerous situations and recommend and apply appropriate buffs based on the user's skill level.

Similar to previous research that ranks user proficiency based on gameplay performance [8], this aims to comprehensively understand the player's state through various indicators.

Compared to existing systems that only predict user skill [11], this system aims to provide personalized support that considers both the situation and the user's level by combining real-time risk assessment (Danger Score) and machine learning-based skill classification (RandomForestClassifier), rather than simple conditional branching or fixed effects.

1) *Risk Assessment and Skill Classification Mechanism:* The main data values used for user class selection from player data stored in Redis are shown in <Table I>.

TABLE II
Key Data Values Used for User Class Selection

Category	Item	Description
Survival Metrics	survivalTime	Total survival time (seconds)
	currentHp	Current health points
	hitCount	Total number of times hit
	totalDamageTaken	Accumulated damage taken
	moveSpeed	Current movement speed
	healSpeed	Healing speed per second
Combat Metrics	atkPower	Current attack power
	atkSpeed	Current attack speed
	killMonsters	Total monsters killed
	kpm	Kills Per Minute
	criticalChance	Critical hit chance (%)
	criticalMultiplier	Critical hit multiplier
	cooldown	Skill cooldown time
Growth/Situation Metrics	defensivePower	Current defensive power
	level	Current player level
	usedHPitems	Number of healing items used

	numEquipments	Number of equipment items held
	numOrnaments	Number of ornamental items held

The server quantifies the player's current risk based on the received data. The Danger Score is calculated by summing all seven items and rounding to two decimal places, serving as a key indicator for detecting dangerous situations. The calculation formula for the Danger Score is shown in <Table II>.

TABLE II
Danger Score Measurement Method

Metric	Formula	Description
d1 : Surrounding Threat	$\text{nearMonsters} * 1.0$	Immediate risk inferred from the number of surrounding monsters
d2 : Health Risk	$(1 - \text{hp_ratio}) * 20$	How far the current health is from maximum health
d3 : Hit Count	$\text{min}(\text{hitCount}, 10) * 1.5$	Recent hit count (up to a maximum of 10 hits reflected)
d4 : Item Consumption	$\text{usedHPitems} * 1.0$	Number of healing items used
d5 : Healing Amount	$(\text{totalHealAmount} / 10) * 0.5$	Degree of crisis response through total health healed
d6 : Kill Speed	$(1 / \text{max}(\text{kpm}, 0.1)) * 0.5$	Considered dangerous if kills per minute are low
d7 : Defense Correction	5 if $\text{defensivePower} < 10$ else 0	Additional correction risk if defensive power is less than 10

RandomForestClassifier was used as the classification model, classifying player skill into 7 levels from 0 to 6. Users with higher scores are assigned to higher classes (Class 6 being the highest), and lower classes are considered novice players.

The buff effects recommended by the AI are linearly adjusted according to this class value. For example, for an attack_up buff, Class 0 receives a 30% increase effect, while Class 6 only receives about a 12% effect.

2) *Dynamic Buff Recommendation Logic*: The buff recommendation logic follows a multi-level priority system. When Danger Score > 40, an invincibility effect is prioritized. If Danger Score > 25 and HP < 20%, health protection (hp_guard) is prioritized.

If a boss is present and conditions are met, a boss buff (boss_buff) or berserk buff is applied. In normal situations, one of the recovery, attack, or movement buffs is randomly recommended, considering cooldowns, probabilities, and classes.

All buffs are subject to a global cooldown and individual cooldowns to prevent duplicate recommendations or excessive intervention.

E. LLM-based Dynamic Quest Generation System

In this study, we implemented a dynamic quest generation system based on Ollama LLM to maintain engagement for novice players and encourage individual achievement.

This system analyzes the player's current game progress and, based on this, generates and suggests personalized quests in real-time. The overall system's data flow consists of communication between the Unity client, the server, and the Ollama LLM.

In the Client → Server stage, the Unity client periodically serializes the player's status information into protobuf format and sends it to the server.

This information includes key gameplay metrics necessary for quest generation, such as health, survival time, movement distance, and equipment status. In the Server → LLM stage, the server constructs dynamic prompts based on the received player data and transmits them to the LLM running on Ollama.

In the LLM → Server → Client stage, the generated quest text is sent back to the client via the server. The server acts as an intermediary without further processing the text. In the Client Parsing and Application stage,

the client analyzes the received natural language text using a pre-defined regular expression-based parser. This extracts structured quest data, such as QuestConditionType, threshold, and targetValue, and registers it in the in-game quest system in real-time.

This structure, unlike existing methods that simply provide fixed content, allows for dynamically composing quests tailored to the player's context, offering strengths in personalization and adaptability. We implemented an innovative system that generates quests in natural language and provides them to the player, leveraging generative agents like those utilizing LLMs [7].

IV. RESEARCH RESULTS

A. Server System and Technology Stack Verification

The system was developed in a Windows 11 environment with Python 3.9.5, implementing a client-server communication structure based on Redis 6.2.3 and Protocol Buffers 3.19.4. AI computations utilized Ollama 0.1.34 and the LLaMA2 Chat 7B model, while RandomForestClassifier was configured with scikit-learn 1.2.2.

The combination of these technologies confirmed the stable provision of real-time data processing and AI analysis functionalities.

B. Dynamic Support System Application Test

To verify that the Unity simulation environment implemented in this study effectively applies the analysis results from the AI server to gameplay, we conducted a dynamic support system application test.

This test aimed to confirm whether the client accurately receives support effects transmitted in real-time by the AI server, assuming the AI server detects dangerous situations for the player, and immediately reflects them in the in-game mechanics.

As a test scenario, we set up a dangerous situation where the player's health rapidly decreased while surrounded by numerous monsters. In this situation, we confirmed the client's response when the AI server transmitted a 'shield generation' effect to the client.



Fig. 2 Dynamic Support System Application Result

As a result of the experiment, the client's dynamic support effect application module immediately increased the shield value in the player data upon receiving the packet.

This change was reflected in real-time on the in-game HUD, as shown in Fig. 2, allowing the player to visually perceive the support effect. This demonstrates that our simulation environment is capable of reflecting AI system support decisions into gameplay without delay.

C. AI Buff Recommendation Application and Effect Verification

To verify that the AI support system's real-time buff recommendation and application were functioning correctly, we recorded log outputs in a real test environment.

The test scenario was set so that the player's class was classified as Class 3 through real-time analysis. At this point, the cooldown_reduction buff was already applied, and the global cooldown was still in progress, as indicated in the output.

```

[ACTIVE BUFFS 상태]
- cooldown_reduction: 종료됨 → 제거 예정
- crit_boost: 남은 시간 10.50초
[INFO] 'cooldown_reduction' 버프가 만료되어 제거되었습니다.

[ACTIVE BUFFS 상태]
- crit_boost: 남은 시간 10.50초
[RESPONSE] 전송 시작
Code: 0
Message: 서포팅 결과 전송 완료
Payload Type: 4
Packet Type: 4
Payload: {'support': {'crit_boost': {'float_value': 0.14, 'duration': 10.5}}}

```

Fig. 3 Buff Expiration and New Buff Application

The system was implemented to display the remaining duration of currently applied buffs in real-time on the HUD, allowing developers and players to verify them. As shown in Fig. 3, when the buff expiration time was reached, the system automatically removed the buff and explicitly noted the removal in the log.

Subsequently, we verified a scenario where the AI recommended a crit_boost buff based on new dangerous situations or conditions, and accordingly, sent a support packet. Dynamically adjusting difficulty in real-time can provide a positive user experience [12].

D. Ollama LLM Quest Generation Performance Evaluation

In this study, to verify the practicality of the Ollama LLM-based dynamic quest generation feature, we evaluated its performance in terms of Latency (response time) and quest suitability. We measured the time from the quest generation request until Ollama LLM returned the quest.

In the measurement environment (Intel i5-12700K CPU, 32GB RAM, NVIDIA GeForce RTX 3070, Ollama Llama 3.3 model), the total delay time from when the Unity client sent a quest generation request to the AI server until the AI server received the quest from Ollama LLM and transmitted it to the client was measured 100 times, and the average value was calculated.

As a result of the measurement, the average Latency for quest generation using Ollama LLM was approximately 1.5 seconds (1500ms). This indicates a sufficiently low delay time for generating and providing quests in real-time during gameplay. Notably, it demonstrated a performance level that does not interrupt player immersion even in games requiring fast responsiveness, such as the Vampire Survivors-like genre.

The characteristic of Ollama running the LLM in a local environment means that network delay is almost non-existent, which is an advantage that allows players to enjoy the game without significant hindrance [6]. The generated quests were suitable for the in-game context and appeared in forms that improved the player's current situation or presented new objectives. This demonstrates the possibility of providing personalized content that reflects the player's individual game progress and skill level, unlike traditional static quest systems. This study presented the design, implementation, and evaluation of TonePick, a mobile application that enables users to generate personalized audiobooks.

V. CONCLUSIONS

In this study, we successfully implemented Continue AI, an AI-based novice support system. It provides personalized real-time support through an integrated system of RandomForestClassifier 7-level skill classification, Danger_score risk detection, and Ollama LLM quest generation. The Unity-Python-Redis architecture and Protocol Buffers communication ensure scalability and real-time capabilities. This is expected to present a new paradigm for game AI support systems and contribute to improving accessibility for novice players.

Future research includes, first, improving prediction accuracy through AI model advancement. This involves integrating deep learning models like LSTM and Transformer to enhance long-term pattern analysis capabilities and pursuing multimodal analysis by linking with biosignals. Second, extending applicability to various game genres. We will validate the versatility of the system by expanding it to RPG, FPS, and strategy games. Third, establishing a cloud-based distributed processing system. We aim to support a large number of concurrent users through a Kubernetes microservice architecture and expand the developer ecosystem by releasing a Unity Asset Store SDK.

ACKNOWLEDGEMENT

This research was supported by the MSIT (Ministry of Science ICT), Korea, under the National Program for Excellence in SW, supervised by the IITP (Institute of Information & Communications Technology Planning & Evaluation) in 2025" (No. 2024-0-00023).

REFERENCES

- [1]. Hannula, R., "*Balancing Randomness in Action Roguelike Game Design*," Bachelor's thesis, Tampere University of Applied Sciences, 2025.
- [2]. R. Hunicke and V. Chapman, "*AI for Dynamic Difficulty Adjustment in Games*", 2004.
- [3]. D. Kennerly, "*Better Game Design Through Data Mining*", 2003.
- [4]. A. Khoo et al., "*FlexBot, Groo, Patton and Hamlet: Research Using Computer Games as a Platform*", 2002.
- [5]. R. Zhao et al., "*NarrativePlay: Interactive Narrative Understanding*", 2023.
- [6]. Duy H. Nguyen & Peter A. Kara, "*The Influence of the Labeling Effect on the Perception of Command Execution Delay in Gaming. Games*", 2025.
- [7]. Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, Michael S. Bernstein, "*Generative Agents: Interactive Simulacra of Human Behavior*", 2023
- [8]. Hee-Dong Chang, "*Estimation Method of User's Gameplay Skill Level through the Performance of Gameplay Status*", 2017.
- [9]. Chang Hoon Park and Jinseok Seo, "*Difficulty Control of a Scrolling-Shooter Game Using Fuzzy Reasoning*", 2017.
- [10]. Justin T. Alexander, John Sear, Andreas Oikonomou, "*An investigation of the effects of game difficulty on player enjoyment*", 2012.
- [11]. Methasit Pengmatchaya, Juggapong Natwichai "*Identifying player skill of Dota 2 using machine learning pipeline*", 2024
- [12]. Sergi Colomer Ferrer "*Adapting Games To Player Taxonomy*", 2024