# RaidEnv: Exploring New Challenges in Automated Content Balancing for Boss Raid Games

Hyeon-Chang Jeon, In-Chang Baek, Cheong-mok Bae, Taehwa Park, Wonsang You, Taegwan Ha, Hoyoun Jung, Jinha Noh, Seungwon Oh, and Kyung-Joong Kim, *Member, IEEE*

***Abstract*—The balance of game content significantly impacts the gaming experience. Unbalanced game content diminishes engagement or increases frustration because of repetitive failure. Although game designers intend to adjust the difficulty of game content, this is a repetitive, labor-intensive, and challenging process, especially for commercial-level games with extensive content. To address this issue, the game research community has explored automated game balancing using artificial intelligence (AI) techniques. However, previous studies have focused on limited game content and did not consider the importance of the generalization ability of play-testing agents when encountering content changes. In this study, we propose RaidEnv, a new game simulator that includes diverse and customizable content for the boss raid scenario in the MMORPG games. In addition, we design two benchmarks for the boss raid scenario that can aid in the practical application of game AI. These benchmarks address two open problems in automatic content balancing (ACB), and we introduce two evaluation metrics to provide guidance for AI in ACB. This novel game research platform expands the frontiers of automatic game balancing problems and offers a framework within a realistic game production pipeline. The open-source environment is available at a GitHub repository.**

***Index Terms*—Boss raid game environment, content generation, game play testing, MMORPG.**

Manuscript received 12 July 2023; revised 10 October 2023; accepted 7 November 2023. Date of publication 28 November 2023; date of current version 17 September 2024. This work was supported by the National Research Foundation of Korea(NRF) funded by the MSIT under Grant 2021R1A4A1030075, in part by Culture, Sports and Tourism R&D Program through the Korea Creative Content Agency grant funded by the Ministry of Culture, Sports and Tourism in 2022 Project Name: Development of artificial intelligence-based game simulation technology to support online game content production, Project R2022020070, in part by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) under Grant 2019-0-01842, and Artificial Intelligence Graduate School Program (GIST) . Recommended by Associate Editor M. Guzdial. *(Hyeon-Chang Jeon and In-Chang Baek contributed equally to this work.) (Corresponding author: Kyung-Joong Kim.)*

Hyeon-Chang Jeon, In-Chang Baek, and Wonsang You are with the Artificial Intelligence Graduate School, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea (e-mail: kevinjeon119@gm.gist.ac.kr; inchang.baek@gm.gist.ac.kr; u.wonsang0514@gm.gist.ac.kr).

Cheong-mok Bae is with Computer Sciences and Engineering, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea (e-mail: cjdahrl@gmail.com).

Taehwa Park, Taegwan Ha, Hoyoun Jung, Jinha Noh, Seungwon Oh, and Kyung-Joong Kim are with the School of Integrated Technology, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea (e-mail: taehwa-p@gm.gist.ac.kr; hataegwan@gm.gist.ac.kr; nenomigami@gm.gist.ac.kr; noah9905@gm.gist.ac.kr; sw980907@gm.gist.ac.kr; kjkim@gist.ac.kr).

Data is available on-line at https://github.com/CILAB-CT-GAME/RaidEnv.
Digital Object Identifier 10.1109/TG.2023.3335399

## I. Introduction

GAME content balancing is a crucial process in the game industry when releasing new game content. Unbalanced game content can cause player dissatisfaction and frustration. To mitigate this issue, game companies employ balancing processes to prevent overpowered content that diminishes the game experience. Prior to content release, game testers evaluate the content balancing and provide reports to the game designer, who then verifies whether the content aligns with their intentions. Even after content release, game designers analyze game log data and readjust the content to achieve rebalancing.

However, several game companies rely on human testers repeatedly playing new game content to measure its difficulty. This approach has limitations as it does not allow testers to acquire sufficient data owing to time constraints. Moreover, the labor-intensive and expensive nature of this process limits the collection of extensive data from game testers. Recently, several scholars have attempted to employ machine learning techniques to test game content in popular puzzle games [1], [2], [3] and card-based real-time strategy games [4], for testing game content.

Automatic content balancing (ACB) is an automated technique that aims to readjust or recombine game content to ensure a balance of the game. Within the game research community, ACB is regarded as a promising solution to address this challenge by leveraging artificial intelligence (AI) players and generators (i.e., balancers) agents as game testers and designers, respectively. In ACB, two repetitive phases are involved in automating game design tasks and quality assurance: 1) generating new game content with machine learning (PCGML) [5] methods and 2) evaluating the content through play testing with an AI player. These automated sequences allow generator models to be trained through extensive trial-and-error iterations, surpassing the limited trials feasible with human testers. The quality of the generated content depends on the AI player's robustness in playing diverse game contents and fairly evaluating them [6]. Similarly, the content generator should possess the ability to generate various contents according to designer-specified requirements.

In this article, we present RaidEnv, a game environment that aims to encompass the features found in commercial-level MMORPG games, with a specific focus on the boss raid scenario, a popular content type in MMORPGs. This highly customizable environment includes machine learning interfaces for play testing and procedural content generation (PCG) agents.

TABLE I
SUMMARY OF THE ENVIRONMENTS FOR CONTENTS GENERATION AND PLAY-TESTING

| **Environment** | **Ref.** | **PCG Problem** | **Multi-agent Problem** | **Balancing Problem** |
|---|---|---|---|---|
| Ludii | [7] | Game Rule | - | - |
| GVGAI | [8] | Game Rule, Map Layout | - | - |
| VizDoom | [9] | Skill, Map Layout | - | - |
| Mario AI Framework | [10] | Map Layout | - | - |
| Hanabi | [11], [12] | - | Ad-hoc Generalization | - |
| Pommerman | [13] | - | High-performance | - |
| Google Research Football | [14] | - | High-performance | - |
| Fever Basketball | [15] | - | High-performance | - |
| SMAC | [16], [17] | - | Environmental Generalization | - |
| Melting Pot | [18] | - | Ad-hoc Generalization | - |
| Neural MMO | [19] | Massive Map Layout | High-performance | - |
| Hearthstone | [20] | Card Deck Building | - | Single-content Balancing (Card) |
| Tower Defense | [21] | Map Layout | - | Multicontent Balancing (Spawn, Unit) |
| Overcooked-AI | [22], [23] | Map Layout | AI-human Coordination | Single-content Balancing (Layout) |
| Pokémon | [24] | Card Deck Building | High-performance | Automatic Team Assembly |
| microRTS | [25], [26] | Skill, Stats | High-performance | Multicontent Balancing (Skill, Stats) |
| **RaidEnv (MMORPG)** | | **Skill, Stats** | **Environmental Generalization** | **Multicontent Balancing (Skill, Stats)** |

Our contribution includes the proposal of two benchmarks applicable to ACB problems. These benchmarks comprise 1) training generalized play-testing agents and 2) controllable content generation. We conduct the benchmarks within RaidEnv, providing deep reinforcement learning (DRL) methods as baselines, in addition to handwritten heuristics. Each benchmark was individually analyzed, and a combined analysis is presented at the end of this article. The major contributions of this study are summarized as follows.

1) We developed RaidEnv, an open-source game simulator that supports extensive customization, facilitating content generation, and play-testing benchmarks.
2) We propose a benchmark for evaluating play-testing agents, emphasizing their generalization ability when encountering content variations.
3) We propose a benchmark for skill content generation, focusing on controllability and diversity aspects.

The rest of this article is organized as follows. In Section II, we review previous studies to compare the features of our environment, covering existing game environments, and balancing scenarios. Section III provides a detailed description of the new game environment, outlining its key features, and components. In Section IV, we present our first benchmark, which focuses on the development of robust play-testing agents. To adequately evaluate the playtested result of the generated content, we propose adjusted test performance metrics considering the varying difficulty in changing content. In Section V, we introduce the second benchmark, the content generation benchmark, which involves skill content generation and utilizes the play-testing agent proposed in the first benchmark. Using agent-based simulations, the utility of the PCG models were evaluated in terms of benchmarking controllability and diversity. Finally, Section VII concludes this article.

## II. Related Work

Table I summarizes the existing game research platforms, and the development purpose of those games is discussed. The literature review was conducted within popular literature databases, i.e., Google Scholar, the IEEE digital library, and ACM digital library. The keywords used to search for existing simulators were "game simulator," "procedural content generation," "multiagent and multiplayer," and "game balancing." In this process, more than 30 papers with detailed environmental specifications and experiments on PCG or multiagent training were reviewed. In addition, previous work on metastudies was also reviewed for PCG [27] taxonomies and multiagent-based gameplaying [28], [29]. For summarizing previous works, we have included papers from only public research platforms because we are proposing an open-source game research framework. Table I summarizes the previous work in three dimensions of criteria: 1) PCG, 2) multiagent, and 3) balancing.

The PCG criterion describes the content types that are employed for content generation. Several game variables with character properties (e.g., skill and stats) [9], [25], [26], map layout [8], [10], [19], [21], [23], card decks [20], [24], and game rules [7], [8] have been investigated. The game rules and character properties are related to the game mechanics and map layout, while the card deck is considered playable content. Skill and stats are the character-related features and the representative methods to encourage role differentiation in multiplayer games.

The multiagent criterion describes the most frequently addressed problem definition in the game environment. The high-performance class [13], [14], [15], [19], [24], [25], [26] represents the environment that addresses the general multiagent problem and the cooperative gameplay, while ad-hoc and environmental generalization are specific subproblems addressed in the specific environment. Ad-hoc generalization [11], [12], [18] is the generation ability of gameplaying, where unseen agents are not considered in the training step; AI–human coordination [23] refers to the agent's ability to adapt to human behavior policy, and environmental generalization [16], [17] refers to the generalization ability to control diverse types of characters using a single policy.

The balancing criterion determines whether the environment addresses the procedural generation or combinational problems for adjusting the game content to satisfy specific game results measured among two or more players. We arranged the corresponding papers with four taxonomies, where the content types are used for balancing. Specifically, we describe the details of these criteria in Section II-A.

Furthermore, a previous study [30] proposed DRL agent play testing in a collective meta MMORPG game. Here, the DRL agents involve two gameplay styles, i.e., craftsman and adventurer, to simulate the in-game economics and assess the inflation phenomenon. This previous study investigated the simulated economics using only play testing agents. In contrast, the current study addresses automated content balancing problems in cooperative combat scenarios.

## A. Automated Content Balancing

In previous studies (see Balancing Problem in Table I), four representative balancing problems have been proposed: automatic team assembly and single/multicontent balancing. Automatic team assembly aims to predict the power of a team [31] and identify overpowering champion combinations [24] in battler games. Single- and multicontent balancing focuses on regulating game object parameters to ensure fair gameplay or intended symmetry. In single-content balancing, the balancer module regulates single-content item (e.g., skill) to meet game results for a particular setting, while in multicontent balancing, the balancer regulates two or more content elements (e.g., skill and stats). Evolutionary methods are employed in [26] to balance units (characters) in real-time strategy (RTS) games. In particular, Sorochan et al. [26] simulated the game with play-testing agents of varying proficiency levels. The proposed environment, i.e., RaidEnv, can support the multicontent balancing of the character's skill and stats; however, only the skill content item was considered in this study.

This study applies the single-content balancing technique in a different game genre, MMORPG. Although our environment supports multicontent balancing for various game elements, such as skills and stats, we demonstrated content generation for a single content to simplify the problem size. Although this research shares similarities with the VGC AI competition [24], it exhibits notable differences. First, the VGC AI competition focused on deck-building rather than altering the content, aiming to determine the possible combinations. Second, they primarily focused on the framework, whereas ours focused on evaluating generated content. Furthermore, this research emphasizes the robustness of the play-testing agents. We propose a balancing study in a cooperative game scenario, specifically a boss raid encompassing various game design components, such as character class and executable skills. We highlight the significance of play-testing agents' generation ability in game-balancing studies. In addition, the proposed platform, which utilizes a popular machine learning framework, had been published as an open-source model for game researchers to generate accessibility for multicontent generation tasks. This platform presents numerous opportunities for exploring novel balancing scenarios among the game research community.

## B. Multiplayer Game AI Environments

Multiplayer game AI environments primarily focus on designing algorithms that determine the behavior of each agent within a game system involving cooperation, competition, or both among two, or more players. Various environments have been proposed to facilitate multiplayer game AI studies for addressing specific problems using multiagent algorithms, as depicted in Table I.

Several multiplayer game environments have concentrated on achieving superhuman-level AI performance in diverse games. For example, the Starcraft Multi-Agent Challenge (SMAC) [16], based on the RTS game *StarCraft II*, offers a wide variety of configurations, such as the number of units, the environmental emphasis on long-term decision-making problem, and partial observation setting. In the sports game genre, Google Research Football (GRF) [14] and the *Fever Basketball* environment [15] aims to develop high-performing agents in sports-related games.

Recently, the research focus has been shifted toward environments that prioritize performance generalization, as observed in Melting Pot [18] and SMAC [16], [17]. Melting Pot focuses on generalizing agent performance when playing against previously unseen agents. It encompasses sequential social dilemmas (SSD), competitive games, and cooperative games. Our study shares similarities with previous work [17] for achieving strong performance in unseen environments. The previous study [17] had extended the environmental features on the top of the original SMAC environment [16] features. However, we focus on developing robust play-testing agents capable of adapting to a broad range of game content. In addition, our environment introduces varying levels of difficulty through changing game content.

## C. Procedural Contents Generation Environments

PCG studies have explored various types of game content generation, including game level layout, game rule generation, and card deck generation. Game level layout generation has been extensively researched in popular video games, such as *Angry Birds* [32], [33], *Super Mario Bros* [34], [35], *Minecraft* [36], and *Overcooked!* [37]. The objective of level generation is to create playable game content that can be explored and completed by players. Game rule generation involves automatically generating the mechanics, dynamics, and constraints of a game. This area of research has been proposed in platforms, such as general video game AI (GVGAI) [8] and Ludii [7], which utilize game description languages. Card deck generation focuses on searching for optimal combinations of game cards and has been studied in representative card simulation games, such as *Hearthstone* [20] and *Pokémon* [8].

To expand the scope of the PCG research to extensive game contents, new customizable game elements must be considered. However, a limited number of studies have focused on generating game content, such as skills and player characteristics. Therefore, this study introduces a novel content type for procedural generation of game and demonstrates the game generation using reinforcement learning. Game features, such as skills, character classes, and items are common across various game genres, with numerous features shared within the same genre.

To ensure the generality of our framework, we extract several features from commercial games and provide them as controllable parameters. In this study, we referred to the user-generated game wiki sites to cover two popular MMORPG games: *World of Warcraft*[1] (*Blizzard Entertainment*, 2004) [38] and *LostArk*[2] (*Smilegate*, 2019) [39] games and we extracted common properties from the characters and skills. These two games utilize their similar game mechanics to implement their game design and we include the shared game variables. Here, we only included the shared features between these two games to enhance the generalizability of the framework. This approach offers new perspectives on generating game content and opens up new possibilities in the game research community.

## III. RaidEnv: The Boss Raid Environment

### A. Boss Raid Scenario

Boss raids are a prevalent form of cooperative multiplayer content in most MMORPG games. The objective of a raid is to defeat a powerful boss within a limited time frame in a dungeon, while playing alongside other players, with the aim of obtaining valuable rewards. Boss raid content is centered around the challenge of overcoming a formidable boss with limited resources and time. Typically, bosses are designed to be stronger than a single player, featuring complex attack patterns that pose a significant challenge in solo encounters. Certain attack patterns necessitate player cooperation, such as spreading out to minimize damage when the boss launches an area attack. This highlights the importance of real-time communication between players. As a fundamental principle in the design of multiplayer games [40], [41], role differentiation is applied to the boss raid content for assigning specific responsibilities to individual players. In the boss raid scenario, role differentiation is achieved by diversifying the skills and abilities of players, with well-defined roles, such as tankers, dealers, and healers being widely standardized. Commercial MMORPG games often offer numerous character roles to enhance the cooperative gameplay experience. For instance, *World of Warcraft* (WoW) game features over 30 specialized roles.

[1][Online]. Available: https://worldofwarcraft.blizzard.com/

[2][Online]. Available: https://www.playlostark.com/en-us

The increasing complexities of role types and game content pose challenges for game designers for predicting the outcomes of game updates and managing the combined effects of various elements. Therefore, the use of ACB techniques becomes crucial for assisting game designers and developers in this complex landscape. However, only a few studies have attempted to apply ACB in the context of boss raids, and existing attempts considered an inadequate number of customizable features to facilitate comprehensive balancing studies. Consequently, the RaidEnv simulator offers a valuable contribution by providing a customization interface that enables the generation of diverse character roles through game simulation. The implementation details of the boss raid scenario are described in the following section.

### B. Contents Definition

To abstract boss raid content, we have categorized the game elements into three components: 1) player class, 2) character statistics, and 3) skill parameters. Fig. 1 provides an overview of the design process for player classes, and the specific details of each component are described as follows.

*Class:* The player class is a defining characteristic that imparts individuality to player characters and assigns distinct roles within the game. The class of a character consists of two key elements: 1) statistics and 2) a set of skills. For example, the Tanker class specializes in withstanding enemy attacks, possessing unique statistics, such as high health points and defensive bonuses compared to other classes. In addition, tankers possess skills that enable them to mitigate incoming damage and protect their allies.

*Statistics (Stats):* Statistics represent the internal attributes of characters in RPGs and are typically represented as numerical values. RPGs, ranging from traditional tabletop games to modern MMORPGs, utilize various statistics to define character attributes. These statistics can be gained permanently or temporarily through character growth, equipment, consumables, or special abilities. Examples of statistics include health/mana points, strength, intelligence, dexterity, critical chances, and more.

*Skill:* While modern games employ different terms, such as abilities, talents, or traits, in our environment, we simplify these concepts into the common notion of skills. Skills are possessed by characters and can be categorized into two types: 1) active skills and 2) passive skills. Active skills are initiated by a character's action, allowing the player to enact decisions regarding which skill to activate from their available set of skills. Conversely, passive skills are latent abilities that have an impact even when not directly activated by the character. Skills can be differentiated by analyzing conditions for activation, target affected, effects produced, and other relevant factors.

### C. Environment Description

The RaidEnv environment is implemented using the Unity platform, and for machine learning training, we utilized the Unity ML-Agents Toolkit [42]. With the RaidEnv environment as a foundation, we have designed several scenarios tailored

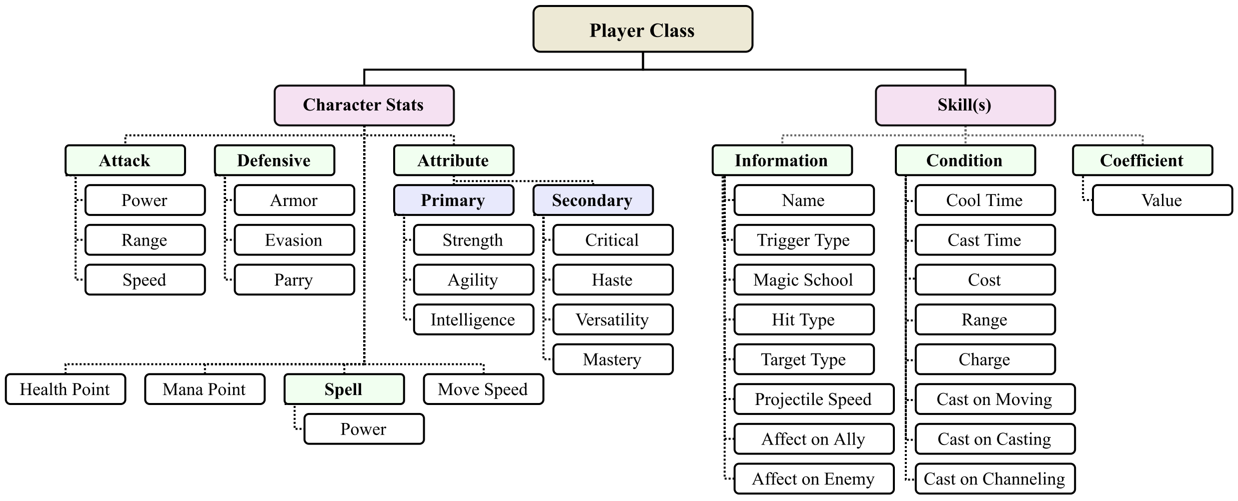

Fig. 1. Overall game components in the RaidEnv environment. The game features and environmental variables are shown in a hierarchical structure, and the equal-level groups are shown in the same color (e.g., light red and light green). Components in white boxes are the modifiable variables, and those is colored boxes are not modifiable (but grouped with similar functional variables). **Character Stats** is a characteristic of the play-testing agents, and **Skill(s)** is a component that can be owned by an agent. Each agent can have one to three executable skills, and all components are fully differentiated by the agent.

to each benchmark. During each training step, the agent collects continuous observations (e.g., scalars and ray perception sensors) and samples discrete actions for learning within the environment. The RaidEnv environment encompasses a basic MMORPG boss raid scenario comprising one enemy agent and three player agents tasked with defeating the boss. Each player agent possesses one skill targeted at the enemy. To efficiently gather play-testing results during training, parallelization of the environment, which is necessary. Therefore, we have included multiple arenas within the environment to enable parallel play testing. Furthermore, we created interfaces to promote learning among multiplayer agents and PCG, including functions to compile game logs for tracing action steps and episodes.

The *Boss* in the boss raid scenario serves as the opponent nonplayer character (NPC). As mentioned in Section III-A, the boss is intentionally designed to be more powerful than a single player to foster cooperation among players. The boss has ten times the health points of a player agent and is equipped with two strong skills with ranges of 6 and 12. The boss properties are configured to design a cooperative boss fight scenario, and the difficulty level is set to barely beat the boss. The boss agent follows a simple behavioral policy: it selects the closest player agent as its target, moves toward that player, and launches an attack when the skill is available. Fig. 2 provides a snapshot of the environment, featuring one boss NPC, and three player agents. In summary, the RaidEnv environment consists of one static boss NPC and three customizable player agents.

### D. Scenario Elements

This article introduces two benchmarks within this framework, as illustrated in Fig. 3. The ACB task can be decomposed into two distinct subproblems: 1) training play-testing agents and 2) generating skill content. Meanwhile, the PCG task necessitates using a robust play-testing agent to evaluate the

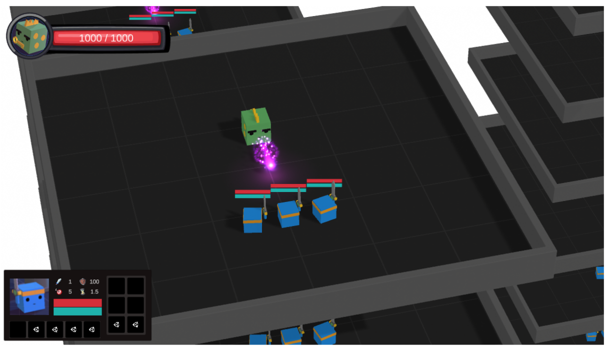

Fig. 2. Snapshot of the RaidEnv environment. The blue ones are the player agents and the green one is the boss agent.

generated content, with one of the play-testing agents serving as the validated tool for the produced skill.

Both the play testing and PCG benchmarks leverage the skill content. In the play-testing benchmark, we utilize the *Range* property to introduce variability in skill characteristics, assessing the agents' ability to generalize across unseen skill ranges. In the PCG benchmark, we incorporate four skill parameters: 1) *Range*, 2) *Value* (damage), 3) *Cast Time*, and 3) *Cool Time* to address the skill generation task. This study incorporates four environment parameters; however, an additional 30 parameters remain to enhance the simulation outcomes, making them more akin to those encountered in commercial games.

## IV. Benchmark 1: Play-Testing Agent With Changing Contents

In this section, we present a play-testing agent trained using multiagent reinforcement learning (MARL) and evaluate its content robustness in different game contents. To effectively utilize the play-testing agent in various scenarios, the agent

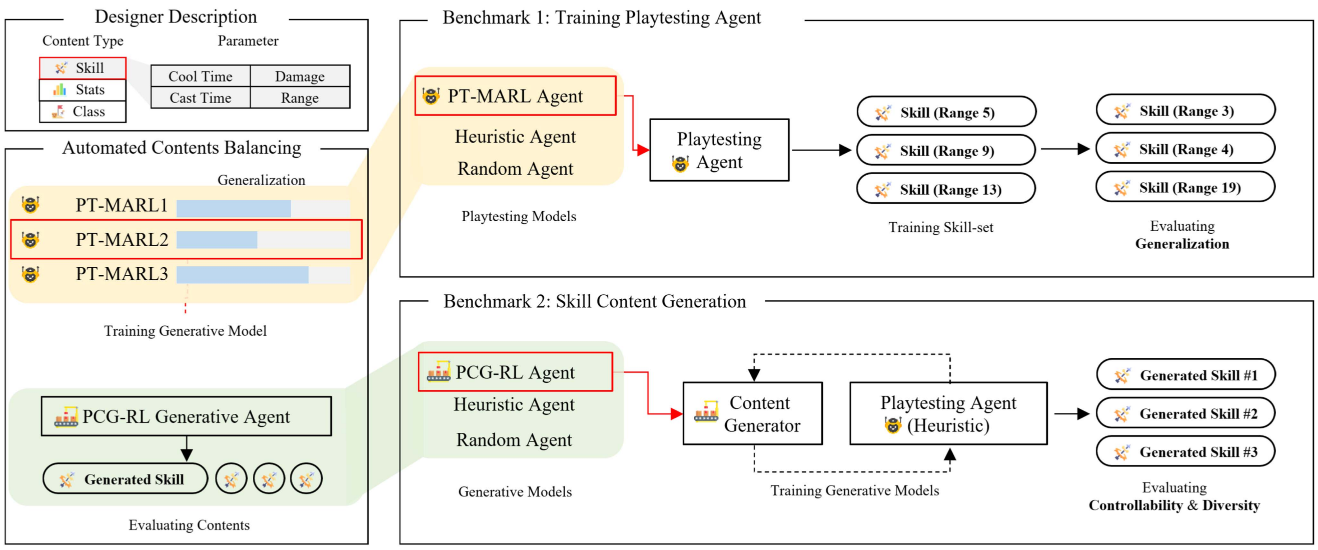

Fig. 3. Proposed two benchmarks in RaidEnv. The overall balancing process includes training play-testing agents and content generation with the play-testing agents process. We separated the two processes as benchmarks to clarify the benchmark tasks. The integrated process (ACB) is considered as future work by integrating the results from the two benchmarks.

should demonstrate adaptability with unseen content parameters. Therefore, we assess whether the agent maintains consistent performance across different skill parameter value settings within the environment. Furthermore, to account for the variation in content difficulty resulting from content changes, we propose the use of a generalization score metric that considers difficulty adjustments using the agent population.

## A. Environment Settings for Play-Testing Agent

*Agent:* Here, we introduce a DRL-based baseline for multiagent gameplaying. The MA-POCA [43] algorithm is a state-of-the-art multiagent algorithm in gameplay. It addresses the posthumous credit assignment problem, which occurs when an agent is removed from the environment during training. This problem is particularly critical in games where agent death is a common occurrence. Moreover, MA-POCA is constructed using the counterfactual multiagent policy gradient (COMA) [44], and incorporates a self-attention layer [45] in the network to handle the posthumous credit assignment problem. The inputs of agents are going throughout the fully connected layer for the embedding and normalization layer as [43]. The objective of MA-POCA involves utilizing a counterfactual baseline [46] and updating for agent $i$ is performed as follows:

$$\mathcal{J}^i_{\text{critic}} = \tau^i - \sum_a Q(s,(a^{-i},a)) \tag{1}$$

$$\mathcal{J}^i_{\text{actor}} = \log\pi_i(a^i|s)A(s,a^i,a^{-i}) \tag{2}$$

$$A(s,a^i,a^{-i}) = G - Q(s,(a^{-i},a^i)) \tag{3}$$

where $\tau^i$ represents the temporal difference (TD) target, $Q$ denotes the state-value function, and $\pi_i$ represents the policy of agent $i$, applicable to both homogeneous and heterogeneous agent settings. $G$ denotes the TD value function estimates.

*State:* At each time step, an agent collects observations on the boss and the status of all team members, including position, velocity, health, and remaining skill cooldown. In addition, the agent gathers information on existing skill projectiles. Furthermore, to facilitate generalized behavior across skill parameters, the agent obtains parameters, such as range, damage, and cast time associated with the skills. Total the number of states size is 33. All features are normalized based on their maximum values.

*Action:* Agents have a total of eight actions, including move, rotate, and execute skills. In our setup, each agent can execute a single action per time step, such as stay, move forward, move backward, turn right, turn left, move left, move right, and execute skill.

*Reward:* Agents receive individual rewards from the environment based on the amount of damage they inflict on the boss. The damage reward is calculated as damage $\times$ 0.01. In addition, agents receive group rewards from the environment, which are set to 1.0 when the boss is defeated. Moreover, we introduce a cooperative element by incorporating a back attack reward. A back attack occurs when one agent distracts the boss while other agents turn around and attack the boss' vulnerable rearside. The reward for successful back attack is calculated as damage $\times$ 0.012, which is higher than the reward for regular attacks.

## B. Play-Testing Agent Evaluation Setup

*1) Baselines:* In the context of RaidEnv, employing play-testing agents capable of handling the complexity of MMORPG games while showcasing robustness to content variations, which is vital for play testing. To validate the robustness of content variations, we employ three play-testing agents: 1) the reinforcement learning agent utilizing the MARL algorithm [43] (PT-MARL), 2) the play-testing heuristic (PT-HR) agent, and 3) the play-testing random (PT-RD) agent.

TABLE II
SKILL RANGE VALUES IN ROBUSTNESS EXPERIMENT

| Agent | #Samples | Sampled Skill Ranges |
|---|---|---|
| PT-MARL1 | 1 | [5] |
| PT-MARL2 | 2 | [5, 9] |
| PT-MARL3 | 3 | [5, 9, 13] |
| PT-MARL4 | 4 | [5, 9, 13, 17] |

1) *PT-MARL* agent makes a decision regarding the actions taken to defeat the boss and receives rewards based on the combat performance. In the self-play experiment, the PT-MARL agent is trained and tested using the same skill range environment. In contrast, we compared the performance of the unseen skill range environment settings. The detailed parameters are listed in Table VIII.
2) *PT-HR* agent follows a simple strategy, i.e., it moves around the boss agent and selects available skills randomly during combat. Here, the agent maintains a maximum attack range from the boss and avoids its attacks. This is the baseline used to compare the multiagent algorithm.
3) 3)*PT-RD* agent selects actions from the action space by uniformly sampling distributions.

*2) Experiment Setup and Evaluation Metrics:* In this benchmark, we conducted two experiments: 1) comparing the heuristic play-testing agent to the MARL agent using fixed content parameter settings and 2) evaluating the agents' robustness with unseen content parameters.

*Optimality:* The experiment was conducted to compare the win rate of the PT-RD, PT-HR, and PT-MARL algorithms across various skill range values, specifically skill ranges of 5, 9, 13, and 17. PT-MARL agents who trained in each skill range evaluated them at the same skill range (e.g., a DRL model was trained on skill 5 and evaluated on the same condition, i.e., skill 5).

*Robustness:* This experiment was conducted to measure the agents' generalizability by varying the number of skill range values. Here, the evaluation environment equips different skill settings with the training environment, and it is assumed as an updated game version. The metric evaluates the availability of the play-testing agent in the updated game environment. Here, the agents were labeled PT-MARL1, PT-MARL2, PT-MARL3, and PT-MARL4 based on the number of skill range values, which the agent was sampled on training. The corresponding values are summarized in Table II. The agent uniformly samples one of the skills and trains an episode with the sampled skills; further, the skill is changed at the beginning of each episode.

The robustness is measured with the performance ratio that increases/decreases in an unseen environment. We adopt the win rate metric to evaluate an agent's performance in an environment, and the win rate is a widely used metric for agent-based play testing [2], [47]; the win rate is calculated by dividing the number of wins by total number of episodes.

To utilize the approach [48] for measuring the performance generalization of environment diversity, we measured the test gap between the heuristic play-testing agent and PT-MARL agents with various skill range parameter values. Here, we measured the performance of a setting, which is commonly found in trained skill range parameter settings, while the content parameter setting that is not found in any agent was selected for the test.

TABLE III
WIN RATE BETWEEN MARL AND HEURISTIC AGENT

| Parameter | $Skill.Range$ | | | |
|---|---|---|---|---|
| Value | 5 | 9 | 13 | 17 |
| PT-MARL | **0.485** (±0.013) | **0.857** (±0.143) | **0.990** (±0.006) | **0.990** (±0.008) |
| PT-HR | 0.089 (±0.017) | 0.371 (±0.025) | 0.698 (±0.021) | 0.830 (±0.089) |
| PT-RD | 0.000 (±0.000) | 0.002 (±0.002) | 0.029 (±0.006) | 0.058 (±0.013) |

The bold text denotes the best (highest) value among the values.

Unlike the previous environment [48], where the game difficulty was constant even if the parameter changes, the proposed approach considers skill range changes that causes variations to the difficulty level. For example, if adjusting a skill's range values leads to reduced attack range, the scenario becomes increasingly difficult. Here, we denote the agent's performance in unseen skill ranges as $\text{score}^{\text{unseen}}$. To calculate an adjusted score, we generated a population ($N = 5$) trained with the unseen skill ranges (3, 4, 19, and 20) and obtained the average performance $\text{score}^{\text{unseen}}_{\text{population}}$ of the population for a representative score in that skill range. Further, we calculated the adjusted score AdjustedScore as follows:

$$\text{AdjustedScore} = \frac{\text{score}^{\text{unseen}}}{\text{score}^{\text{unseen}}_{\text{population}}}. \tag{4}$$

## C. Experimental Result

*1) Optimality:* In this section, we present a comparison of the average win rate among the MARL agent (PT-MARL), PT-HR agent, and PT-RD agent over 500 games. The objective is to demonstrate why RL agents can be used for obtaining a more reliable test result. The PT-MARL agent was trained with 50 million steps for every five runs according to the skill range setting. As shown in Table III, the PT-MARL agents exhibited better results than the PT-HR and PT-RD in their corresponding settings regardless of the range. Specifically, the PT-HR agent demonstrated worse results as the skill range decreased . This indicates that the PT-HR has limitations for solving difficult levels of the game, while the complex PT-MARL agent solves it well. The PT-RD agent cannot win even one game in the range 5 because the game is much more difficult because the agents should get close to the boss. In contrast, as the range increases, even simple PT-RD agents can defeat the boss, suggesting that the game's difficulty is low. The PT-MARL agent converged successfully, unlike the relatively fast convergence of skill ranges 13 and 17 settings; skill ranges 5 and 9 settings require more steps for convergence during training, as shown in Section IV.

*2) Robustness:* We trained the four MARL models (i.e., PT-MARL1, -MARL2, -MARL3, and -MARL4) using 50 M steps with the four skill sets (Table II), and the performance was averaged with five seeds. After the training, the unseen skill ranges 3, 4, 19, and 20 were evaluated and compared. The results

TABLE IV
TESTING AND TRAINING PERFORMANCE WITH ADJUSTED DIFFICULTY LEVEL

| | Train | Test | | | | |
|---|---|---|---|---|---|---|
| | 5 | 3(Hard) | 4(Hard) | 19(Easy) | 20(Easy) | Test Avg. |
| PT-MARL 1 | **1.049** (±0.063) | 0.004 (±0.005) | **0.553** (±0.085) | 0.597 (±0.055) | 0.597 (±0.048) | 0.506 (±0.310) |
| PT-MARL 2 | 1.004 (±0.122) | **0.028** (±0.033) | **0.553** (±0.224) | 0.523 (±0.082) | 0.525 (±0.079) | 0.473 (±0.306) |
| PT-MARL 3 | 0.630 (±0.268) | 0.013 (±0.024) | 0.369 (±0.192) | 0.826 (±0.109) | 0.778 (±0.151) | **0.511** (±0.351) |
| PT-MARL 4 | 0.049 (±0.031) | 0.000 (±0.000) | 0.016 (±0.015) | **0.980** (±0.017) | **0.971** (±0.022) | 0.443 (±0.477) |
| PT-HR | 0.119 (±0.038) | 0.013 (±0.007) | 0.115 (±0.015) | 0.89 (±0.011) | 0.868 (±0.021) | 0.441 (±0.396) |

The bold text denotes the best (highest) value among the values.

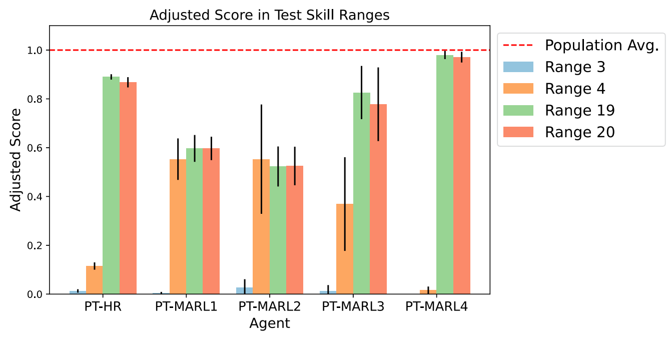

Fig. 4. Figure of adjusted score results in test skill range settings. The red line indicates the average adjusted score of the population ($N = 5$) in unseen settings. Adjusted score is the ratio of the test performance of the trained agent to the average performance of the population. As the adjusted score achieves a higher test adjusted score, the score is near 1 (sometimes even higher than 1).

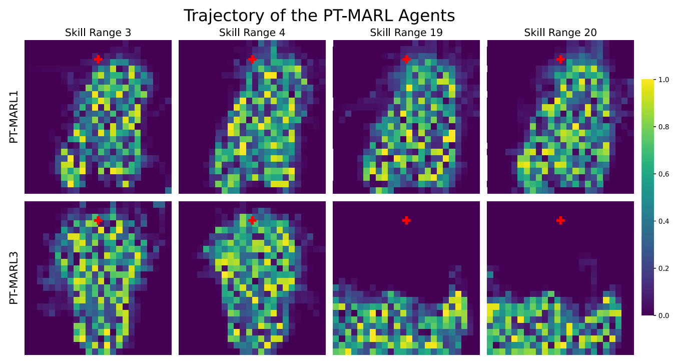

Fig. 5. Heatmap of the average game position occupancy of PT-MARL1, and PT-MARL3 using 500 games. The red spot is the initial position of the boss. The results show that the PT-MARL1 agent did not change strategy effectively even when the skill range changed. In contrast, the PT-MARL3 agent recognized the change in a parameter, which enabled it to change its movement strategy.

are described in Table IV and Fig. 4. Fig. 4 demonstrates the comparison between the trained agents and nonlearning PT-HR agents. The population of agents ($N = 5$) serves as an indicator of game difficulty at each skill range. Notably, this population was established by training a separate MARL algorithm for each skill range. The performance of the population is shown in Table VII in Appendix.

Overall, PT-MARL agents outperformed PT-HR agents in the test content parameter settings. Among the PT-MARL agents, PT-MARL3 exhibited the highest average performance for the test skill ranges (3, 4, 19, and 20), as in Table IV. In contrast, PT-MARL4 displayed intensive learning of long-distance skills but struggled with short-distance encounters. PT-MARL1 demonstrated consistent performance across all test content parameter settings because of its reliance on a single strategy. Notably, all agents struggled to learn in the skill range 3, suggesting room for further improvement and development. See Table VII for detailed information regarding the adjusted score and raw information.

## D. Further Analysis

We analyzed the movement patterns of PT-MARL1 and PT-MARL3 agents based on the number of sampled skill ranges. We played 500 games for each agent and averaged their occupied locations. As depicted in Fig. 5, the PT-MARL1 agent, who only observed a skill range of 5, exhibited consistent behavior regardless of skill range changes. In contrast, the PT-MARL3 agent recognized the variation in skill ranges and adapted different strategies accordingly. For example, PT-MARL3 agents move to the boss and shoot skills in skill range 4. In contrast,

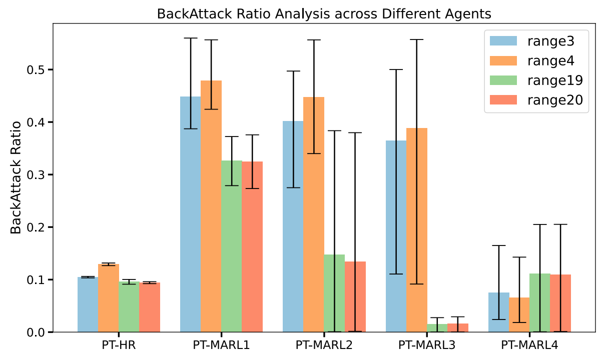

Fig. 6. Back attack ratio in test skill range settings. The back attack ratio counts the number of back attacks and divides it by the total number of attacks. Here, the black line is the minimum and maximum of the back attack ratio among five runs.

the PT-MARL3 agents executed skills sustaining the maximum range of the skills when they had a higher skill range (skill ranges 19 and 20).

Fig. 6 compares the back attack ratios of PT-HR, PT-MARL, PT-MARL2, PT-MARL3, and PT-MARL4 in unseen skill ranges for over 500 games. For PT-HR, a back attack rate of approximately 10% was obtained in the test ranges. On average, the PT-MARL1 agent exhibited the best performance in terms of the back attack ratio. Because PT-MARL1's strategy is not changed (Fig. 5), PT-MARL1 did not exhibit many variations for the back attack ratio across the skill range. Although the PT-MARL3 agent achieved the highest adjusted score on average, it did not exhibit a significant increase in back attacks in skill ranges 19 and 20.

## V. Benchmark 2: Contents Generation

In this section, we present three baselines for the content generation task in the boss raid scenario. The primary objective of designing PCG agents is to generate or rebalance game content to achieve specific target game outcomes. For instance, a game designer may request the generator agent to produce a skill that yields a desired win rate (e.g., 60%), and the agent should adjust the skill parameters to achieve the designated win rate. This approach can be applied to both generating new game content from scratch and rebalancing existing content. We introduce two notations for this task: 1) current win rate, $W_c$, which represents the simulated result with arbitrary game content, and 2) target win rate, $W_t$, which is the desired outcome of the generation process. The ultimate goal of this task is to minimize the difference between $W_c$ and $W_t$ by adjusting the skill parameter values.

As an example of skill content generation, we propose a DRL method. The DRL framework for PCG was initially introduced in [49], and the authors of [50] further enhanced this framework to improve its controllability. As a machine learning approach for content generation, we adopt a DRL-based baseline utilizing the controllable PCGRL framework presented in the previous study [49]. The specific details of the DRL implementation for our work are described as follows. To evaluate the generated content, we employ the heuristic play-testing agent discussed in the previous section (Section IV-B), and we refer to this agent as the PT-HR agent to avoid confusion with the heuristic PCG agent.

### A. Environment Settings for DRL

*Agent* PCGRL (PCG via RL) [49] proposed a new framework for generating game content by training a DRL-based generative model. The state is defined as the scalar or tabular representation of game content; the action involves modifying the content; while the reward is measured based on the completeness of the generated content. Subsequent works, such as controllable PCGRL have been introduced for 2-D [50] and 3-D games [36] by designing goal-oriented reward functions that calculate the distance to a designer-defined value. Controllable PCGRL facilitates the training of a generative model to produce a desired output specified by the designer, enhancing the utility of PCGRL models by enriching customization features and reducing quality assurance labor. In this study, we utilize controllable PCGRL to demonstrate the skill generation task in Section V, training the generative model to generate diverse game outcomes.

*State:* The state for the generator includes the skill parameter values required for generating the target skill. The generator agent receives four scaled values representing the balancing parameters: cool time, range, damage, and cast time. In our setup, each value is scaled between a minimum and maximum value. For example, cool time and range is defined as 0.5–0.6, the skill range is set from 1–20, the damage range is between 0.5 and 1, and the cast time falls within the interval of 0.5–1.5.

*Action:* The generator has an action space of $N$, where each action consists of five discrete values. Here, $N$ represents the number of skills, and the action determines the parameter updates. We consider four parameters for skill generation ($N = 4$): cool time, range, damage, and cast time. The action determines the increment or decrement of each parameter using a handcrafted scaler. The granularity of the scaler was set to 0.16% of each parameter range allowing for decreases (-) or increases (+) of [-0.16%, -0.08%, 0.0%, +0.08%, +0.16%] within each parameter range.

*Reward:* The reward function is designed to provide nonsparse reward signals to the generator. The generator receives positive rewards when it updates the skill parameters to achieve the target game win rate ($g$, goal). To design the reward function for a controllable generator, we adopt a well-designed reward system proposed in the PCGRL framework [36], [50]. The L1 norm distance to the goal $l$ is calculated as $l_t = \text{abs}(||g_t - p_t||_{L1})$, where $p_t$ represents the playtested win rate at time $t$. The reward at time $t$ is then calculated as $r_t = l_{t-1} - l_t$. A positive reward value indicates that the updated skill is closer to the target win rate. In summary, the generator was trained to adjust the skill parameters for achieving the designer-defined play-testing result.

Notably, $p$ is measured by simulating the generated skills multiple times under the same conditions. In our setup, we repeat the simulation ($N = 100$) using the PT-HR agent described in Section IV-B, and we report the average value of $p$ in the experimental results. The use of the PT-HR agent is preferred in PCG studies due to its reasonable performance and lower computational cost.

*Terminal:* The episode was terminated on the simulated win rate that is sufficiently close to the target game win rate, and the difference between the play-tested result and the target win rate was $< 0.02$, i.e., $\text{abs}(||g_t - p_t||_{L1}) < 0.02$. The maximum episode length was 40, and the environment was reset once the termination condition was satisfied. We terminated the episode if the content generation is completed to enhance the sampling for training the DRL agent and this condition was used only in the training step.

### B. Generator Evaluation Setup

*1) Baselines:* We compare the performance of three content generator models in the content generation experiment: the proposed PCG agent utilizing the proximal policy optimization (PPO) algorithm [51], PCG-heuristic (PCG-HR) agent, and PCG-random (PCG-RD) agent. PPO is a popular DRL algorithm known for its stability and has been widely used in previous content generation studies [36], [49]. To provide performance comparisons for the PPO agent, we include a simple heuristic agent and a random agent.

1) *PCG-RL* agent is based on the DRL approach and is implemented using the PCGRL framework [49], as detailed in Section V-A. The PPO agent enacts decisions to update skill parameters and receives rewards based on the play-testing results. The detailed parameters are shown in Table IX.

2) *PCG-HR* agent greedily adjusts the skill parameters based on the expert knowledge. The agent selects one of the four skill parameters and adjusts the selected parameter based on the play-tested win rate result, which is returned from the environment; the parameter is adjusted to meet the target win rate. The agent increases or decreases one of the skill parameters per action (e.g., the agent reduces the cool time value if the play-tested win rate is smaller than the target win rate.).
3) *PCG-RD* agent adjusts the parameters without considering the target win rate. The agent randomly samples one action from the five possible action branches in uniform probability. Each random action is then applied to the four skill parameters.

*2) Experiment Setup:* We evaluated the performance of the generator models, PCG-RL, PCG-HR, and PCG-RD. We compiled 300 game results at each step to ensure an accurate evaluation of the generated contents. The heuristic play-testing agent (PT-HR in Section IV-B) was employed during the play-testing process. In addition, this section focuses on demonstrating the generation of skill contents, and the heuristic method speeds up the experiment time and enhances the reliability of the evaluation.

Because PCG-RL is trained from a randomly initialized neural network, we repeated the experiment five times to mitigate performance variations caused by initial weights. After training the PCG-RL models for 20K steps, we generated 1K skills for each model, resulting in a total of 5K samples (5 models $\times$ 1K samples $=$ 5K samples). All experimental results are summarized by averaging the simulated game results across the five seed runs.

*3) Evaluation Metrics:* To evaluate the contents generated by the generator models, two commonly used metrics in PCG studies are used: 1) controllability and 2) diversity. A lack of controllability implies that the agent cannot generate contents that fulfill the game designer's requirements, while a lack of diversity results in repetitive content that can only be used once.

*Controllability:* Controllability is measured by evaluating the error between the designer's desired target win rate ($W_t$) and the simulated results obtained from the generated skills ($W_c$). The error measurement for controllability is defined in (5), where $N$ represents the number of samples

$$\text{WinrateError} = \sum_{i}^{N} |W_{i,t} - W_{i,c}|. \tag{5}$$

$W_t$ denotes the target win rate desired by the designer, and $W_c$ represents the win rate of the generated skill. The equation calculates the mean error of the win rates of skills relative to the target value. The lower error indicates that the skills produced by the generator are more closely aligned with the designer's requirements.

*Diversity:* Diversity measures the variety of styles in which the contents are generated while still satisfying the intended conditions. A higher diversity enriches the gameplay experience and provides designers with a wide range of options. The procedure for sampling skills is the same as in controllability; however, we apply a root mean squared error (RMSE) $< 0.1$ filter to evaluate the diversity of the contents. This filtering ensures that the evaluated diversity that represents the skill parameter ranges satisfies the designer-defined win rates; the diversity metric measures the diversity of the generated content that meets a target play-testing result (e.g., win rate). Therefore, diversity assesses the range of choices in the options, where the generator provides the designer with numerous options depending on the designer's desired conditions. A higher value indicates that the generator has generated contents with diverse skill parameter ranges, resulting in various game experiences even with the same balanced results ($W_t$). To simplify the interpretation of the skill diversity, we employed the principal component analysis (PCA) [52] to represent the four parameters as a single value.

TABLE V
DESCRIPTIVE STATISTICS ON THE CONTROLLABILITY OF THE GENERATIVE MODELS

| Target ($W_t$) | Method | Generated ($W_c$) Mean ($\pm$SD) | RMSE ($\|W_t - W_c\|$) Mean ($\pm$SD) |
|---|---|---|---|
| 0.1 | PCG-HR | 0.107 ($\pm$0.065) | **0.044 ($\pm$0.049)** |
| | PCG-RL | 0.149 ($\pm$0.242) | 0.156 ($\pm$0.191) |
| | PCG-RD | 0.354 ($\pm$0.389) | 0.335 ($\pm$0.321) |
| 0.2 | PCG-HR | 0.206 ($\pm$0.098) | **0.074 ($\pm$0.065)** |
| | PCG-RL | 0.266 ($\pm$0.343) | 0.268 ($\pm$0.225) |
| | PCG-RD | 0.366 ($\pm$0.381) | 0.333 ($\pm$0.249) |
| 0.3 | PCG-HR | 0.297 ($\pm$0.108) | **0.079 ($\pm$0.073)** |
| | PCG-RL | 0.217 ($\pm$0.189) | 0.169 ($\pm$0.119) |
| | PCG-RD | 0.418 ($\pm$0.382) | 0.348 ($\pm$0.196) |
| 0.4 | PCG-HR | 0.416 ($\pm$0.122) | **0.089 ($\pm$0.085)** |
| | PCG-RL | 0.377 ($\pm$0.315) | 0.276 ($\pm$0.153) |
| | PCG-RD | 0.381 ($\pm$0.394) | 0.362 ($\pm$0.154) |
| 0.5 | PCG-HR | 0.489 ($\pm$0.133) | **0.099 ($\pm$0.089)** |
| | PCG-RL | 0.435 ($\pm$0.334) | 0.296 ($\pm$0.167) |
| | PCG-RD | 0.390 ($\pm$0.391) | 0.379 ($\pm$0.144) |
| 0.6 | PCG-HR | 0.592 ($\pm$0.132) | **0.097 ($\pm$0.090)** |
| | PCG-RL | 0.606 ($\pm$0.309) | 0.276 ($\pm$0.138) |
| | PCG-RD | 0.409 ($\pm$0.395) | 0.405 ($\pm$0.168) |
| 0.7 | PCG-HR | 0.684 ($\pm$0.113) | **0.088 ($\pm$0.073)** |
| | PCG-RL | 0.652 ($\pm$0.354) | 0.292 ($\pm$0.205) |
| | PCG-RD | 0.387 ($\pm$0.397) | 0.445 ($\pm$0.239) |
| Mean | PCG-HR | - | **0.081 ($\pm$0.078)** |
| | PCG-RL | - | 0.248 ($\pm$0.183) |
| | PCG-RD | - | 0.373 ($\pm$0.221) |

The bold text denotes the best (highest) value among the values.

## C. Experimental Result

*1) Controllability:* The descriptive results for measuring the controllability of the generated win rate targets ($W_t$) are listed in Table V. Cells with a blue background color indicate lower values, while red cells indicate higher values. We trained the PCG-RL model using seven target parameters ranging from 0.1–0.7 and compared its performance with the mentioned approaches. Each of the three models generated $N = 1000$ skill parameters for each target ($W_t$) and was evaluated using the heuristic play-testing agent (PT-HR).

TABLE VI
DESCRIPTIVE STATISTICS ON THE DIVERSITY OF THE GENERATED CONTENTS

| Target ($W_t$) | Method | Range (±SD) | Cool Time (±SD) | Cast Time (±SD) | Damage (±SD) | PCA (±SD) |
|---|---|---|---|---|---|---|
| 0.1 | PCG-HR | 0.258 | 0.300 | 0.269 | 0.265 | 0.302 |
| | PCG-RL | **0.315** | **0.482** | 0.258 | **0.472** | **0.625** |
| | PCG-RD | 0.266 | 0.325 | **0.294** | 0.283 | 0.336 |
| 0.2 | PCG-HR | 0.262 | 0.324 | 0.249 | 0.288 | 0.298 |
| | PCG-RL | **0.351** | **0.381** | **0.292** | **0.428** | **0.421** |
| | PCG-RD | 0.284 | 0.329 | 0.279 | 0.316 | 0.359 |
| 0.3 | PCG-HR | **0.271** | 0.312 | **0.265** | 0.292 | 0.312 |
| | PCG-RL | 0.218 | **0.398** | 0.261 | **0.349** | **0.488** |
| | PCG-RD | 0.268 | 0.331 | 0.264 | 0.308 | 0.329 |
| 0.4 | PCG-HR | 0.252 | 0.312 | 0.233 | 0.278 | 0.304 |
| | PCG-RL | 0.180 | **0.377** | **0.364** | **0.476** | **0.559** |
| | PCG-RD | **0.277** | 0.333 | 0.287 | 0.292 | 0.358 |
| 0.5 | PCG-HR | 0.257 | 0.316 | 0.236 | 0.269 | 0.310 |
| | PCG-RL | 0.204 | **0.393** | **0.391** | **0.405** | **0.519** |
| | PCG-RD | **0.278** | 0.328 | 0.280 | 0.269 | 0.326 |
| 0.6 | PCG-HR | 0.247 | 0.313 | 0.236 | 0.269 | 0.303 |
| | PCG-RL | 0.105 | 0.284 | 0.214 | 0.212 | 0.258 |
| | PCG-RD | **0.261** | **0.317** | **0.302** | **0.306** | **0.385** |
| 0.7 | PCG-HR | 0.257 | 0.293 | 0.227 | 0.263 | 0.293 |
| | PCG-RL | 0.194 | 0.271 | **0.447** | **0.406** | **0.625** |
| | PCG-RD | **0.262** | **0.322** | 0.258 | 0.305 | 0.358 |
| Mean | PCG-HR | 0.258 | 0.310 | 0.246 | 0.275 | 0.303 |
| | PCG-RL | 0.237 | **0.375** | **0.328** | **0.402** | **0.513** |
| | PCG-RD | **0.271** | 0.326 | 0.281 | 0.297 | 0.351 |

The bold text denotes the best (highest) value among the values.

According to this criterion, the PCG-HR agent demonstrated the best performance, followed by the PCG-RL agent in second place, and the PCG-RD agent in last place. The PCG-HR agent consistently achieved the lowest RMSE values across all target win rate conditions. On average, its error in win rate estimation was approximately 8.1%. Surprisingly, the PCG-RL agent performed worse than the heuristic agent. However, we observed that the PCG-RL model was trained to minimize win rate errors compared to the random agent. We discuss the reason for the low performance of the PCG-RL agent in the discussion section (Section VI-B).

*2) Diversity:* Table VI presents the descriptive results for measuring the diversity of the generated content characteristics. Different background colors represent each parameter type, with darker colors indicating higher values. We provide the standard deviation (SD) values for each skill parameter in separate columns of Table VI. The SD value is calculated using $N = 300$ samples for each parameter, and the samples are filtered based on a threshold (RMSE$< 0.1$) to measuring the diversity using only with valid skills that meet the designer's requirements. A higher SD value indicates that the generator generated contents with diverse skill parameter ranges, resulting in various game experiences even with the same balanced results ($W_t$). To better understand the diversity of the skills, which are represented in 4-D values (i.e., range, cool time, cast time, and damage), we employed the PCA [53] technique, a dimension reduction technique, to represent the four parameters in a 1-D metric. By representing the skills in this metric, we measured the variance of the skills using a single value that encompasses all the parameters of the skill content. The mean SD value is calculated across all win rates. The best values are marked in bold, and PCG-RL outperformed the other models in diversity, showing the best performance in six out of seven conditions. This result implies that the PCG-RL agent can provide rich options for designers even when given a specific condition. We conducted a further analysis of the diversity using a visualization technique, which is discussed in Section V-D.

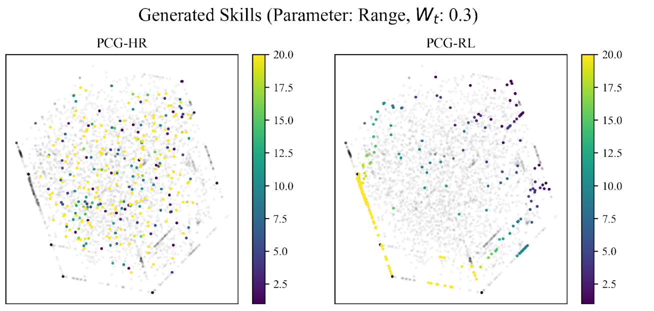

Fig. 7. Visualization of generated skills in a 2-D plot using t-SNE, based on range parameter. Totally, 2100 entries (300 samples $\times$ 7 target win rates) are involved, respectively. Note that gray dots denote all skills generated by the PCG-HR and -RL, and colorized dots denote the skill generated with each generator.

## D. Further Analysis

In diversity criteria (Section V-C2), the PCG-RL agent outperformed other methods, while showing the highest variance in generated skill parameters. Fig. 7 illustrates visualizations of the skills generated by PCG-HR and PCG-RL agents; the points denote the skills, and the 4-D parameters were reduced to two dimensions using the PCA dimension reduction technique. We employed the skills generated with a target win rate of 0.3 ($W_t = 0.3$) parameter to clarify the visualization, and colorization was applied using the range parameter values. The skills generated by the PCG-HR are mostly located at the center of the plot; meanwhile, the skills generated by PCG-RL are more widely dispersed than those generated by PCG-HR. This clue suggests that the PGG-RL generated more conspicuous characteristic skills than PCG-HR. The distinguished skills induce high variances in the diversity metric (PCA column in Table VI). Interestingly, the colors that indicate the range value are congregated with the same color in the same cluster in the PCG-RL plot. This indicates that the skills within a cluster have similar skill parameter values; further, these characteristics exhibit little diversity within each cluster. Consequently, the clues imply that the PCG-RL could generate more distinguishable skills; however, the diversity in a distinguished skill group could be low. We discuss the low diversity in a distinguished skill cluster in Section VI-B.

## VI. DISCUSSION

### A. Play-Testing Agent With Changing Contents

Although MARL algorithms have demonstrated successful learning across all content difficulties, a fully robust agent has yet to be achieved. Specifically, PT-MARL4 encountered difficulties in learning even with seen skill range 5 (see Table IV). This suggests that the PT-MARL4 agent struggled to adapt to challenging environments due to overfitting caused by its ease in learning from simpler environments. Furthermore, in the case of the skill range 3, representing extremely challenging levels, the agent failed to generalize altogether. Nevertheless, the PT-MARL agent showcases its strength in slightly more challenging environments. We anticipate that learning can be accelerated through approaches, such as curriculum learning.

### B. Content Generation

In terms of the controllability criteria (Section V-C1), the DRL-based (PCG-RL) agent underperformed compared to the heuristic (PCG-HR) agent. The suboptimal performance of the PCG-RL agent could be attributed to noise in the reward signal. The reward of the PCG agent is measured via subtracting the previous play-testing results from the current results. The variance of the results is high even with the same skill parameters. The reward noise occurred during the subtraction of the previous game results from the current results and the accumulated error increases due to high variance. This problem disrupts the DRL model training and can explain the low performance compared to PCG-HR. In addition, PCG-HR makes the decision based on a single game result (i.e., the current win rate), and the error of the playtested result is relatively lower than the measuring of PCG-RL reward. Thus, denoising of the simulated result is expected to enhance the performance of the PCG-RL agent. Constructing a world model on game simulation or increasing the repetition of the simulation could address this problem.

In the further analysis of the diversity criteria (Section V-D), the PCG-RL successfully generated distinguished skills; however, the low diversity was observed in a distinguished skill set. DRL attempts to determine an optimal solution rather than diversity, and it can be converged to explore particular skills. Rather than seeking a single optimal solution, an alternative approach is required to encourage the algorithm to alleviate the repetitive exploration of local optimal. In addition, further studies are necessary to measure the diversity in a more empirical manner, where practical game experiences are also considered.

## VII. Conclusion

This article presents the introduction of two benchmarks, namely, training play-testing agents and content generators, to facilitate the ACB technique development. To illustrate ACB subtechnique, a boss raid game environment was developed to support the multiplayer play testing and content generator interfaces. In addition, to extend the applicability of our work to commercial games, various representative game components were examined and integrated as customizable elements in the boss raid game environment. The findings from the two benchmarks are summarized as follows.

In the play-testing benchmark (Section IV), the comprehensive results showed that the MARL agent exhibits superior performance compared to the heuristic agent. Interestingly, no single agent exhibited dominance in terms of generalization, even after training with different skill sets. However, the PT-MARL3 agent exhibited consistent performance across the entire test skill sets, suggesting that a nonoverfitted agent can reliably evaluate newly generated contents. Hence, training a dominant agent capable of performing well with all unseen content remains a challenging problem.

In the PCG benchmark (Section V), the results demonstrated that the DRL-based (PCG-RL) approach achieved intermediate performance between the heuristic and random agents. Although the DRL agent outperformed the other agents in terms of generating diverse content, it exhibited lower controllability compared to the heuristic agent. Content reliability is of the utmost importance because diversity is also measured with a valid skill that satisfies a condition. Therefore, enhancing the controllability of the DRL agents will be the focus of future work.

In conclusion, the development of these two benchmarks contributes remarkably to the advancement of the ACB technique. While integrated experiments were not included in this study, we posit that generalized multiplayer agents can provide reliable reward signals to the PCG agent. In addition, the multiplayer play-testing results, which emulate human-like behavior, contribute to the generation of high-quality content and assist in evaluating human-designed content.

The proposed framework offers numerous avenues for future work in terms of PCG and MARL, which were not explored in this study. In the following, we summarize the relevant limitations of this study. First, we acknowledge the limitation of content types generated in this work. Herein, the content generators only handled four representative parameters; however, there are 13 remaining unused skill parameters, in addition to character stats (see Fig. 1). The reason for using only four representative parameters is to reduce the action shape complexity for the DRL-based approach to present a reasonable problem size for introducing a new content generation. The content generation result that the DRL method exhibited lower performance than the heuristic; hence, there is room for improvement. To handle more diverse content types, future work will consider multiobjective learning [54] is expected to handle multiple reward signals by enlarging the criteria of the play-testing targets.

Second, training robust agents in heterogeneous settings remains a task for future exploration. This would indicate the class (e.g., magician, attacker, or tanker) and type of the player. In this work, we employed a homogeneous agent setting where the characteristics and skills are the same for all agents during play testing. However, in real boss raid scenarios, the performance of play-testing agents may decrease when their teammates change. Therefore, the issue of generalization with respect to different teammates will be a substantial challenge.

## Appendix

### A. Play-Testing Agent Result

Table VII showed the performance on train and test content parameters. The results showed that PT-MARL4 and PT-MARL3 are not trained perfectly using trained settings (seen). Especially, PT-MARL4 has shown that it has failed to learn about the strategy when their skill has short distance. Overall, PT-MARL3

TABLE VII
TEST AND TRAIN ORIGIN PERFORMANCE

| | Train | Test | | | |
|---|---|---|---|---|---|
| | 5 | 3(Hard) | 4(Hard) | 19(Easy) | 20(Easy) |
| Population | 0.45 | 0.39 | 0.44 | 0.992 | 0.995 |
| PT-MARL1 | 0.485 | 0.002 | 0.243 | 0.593 | 0.594 |
| PT-MARL2 | 0.452 | 0.01 | 0.243 | 0.519 | 0.522 |
| PT-MARL3 | 0.284 | 0.05 | 0.162 | 0.82 | 0.774 |
| PT-MARL4 | 0.021 | 0.000 | 0.007 | 0.972 | 0.996 |
| PT-HR | 0.089 | 0.005 | 0.05 | 0.883 | 0.863 |

TABLE VIII
MODEL HYPERPARAMETERS AND EXPERIMENTAL SETTINGS FOR TRAINING MA-POCA

| **Parameter** | **Value** |
|---|---|
| **MA-POCA Agent Setting** | |
| $\lambda$ | 0.95 |
| $\beta$ | 0.01 |
| Number of Epochs | 3 |
| Minibatch size | 256 |
| Clipping coefficient ($\epsilon$) | 0.2 |
| Learning rate | 0.0003 |
| Learning rate schedule | constant |
| **Network** | |
| Hidden Units | 256 |
| Number of layers | 2 |
| **Experimental Setting** | |
| $\gamma$ | 0.99 |
| State Vector Size | 33 |
| Action Vector Size | 8 |
| Goal | extrinsic |
| Maximum steps | 20,000,000 |

TABLE IX
MODEL HYPERPARAMETERS AND EXPERIMENTAL SETTINGS FOR TRAINING PPO

| **Parameter** | **Value** |
|---|---|
| **PPO Agent Setting** | |
| $\lambda_{\text{GAE}}$ | 0.99 |
| Number of Epochs | 3 |
| Rollout Length | 120 |
| Minibatch size | 16 |
| Clipping coefficient ($\epsilon$) | 0.2 |
| Learning rate | 0.0003 |
| Learning rate schedule | constant |
| Entropy Coefficient | 0.005 |
| **Network** | |
| Hidden Units | 256 |
| Number of layers | 4 |
| Return normalization | yes |
| Deterministic | yes |
| **Experimental Setting** | |
| $\gamma$ | 0.99 |
| State Vector Size | 44 |
| Action Vector Size | 4 |
| Goal | extrinsic |
| Reward | L1 norm |
| Maximum steps | 20,000 |

showed the best average performance in unseen environments. However, because all agents fail to generalize in skill range 3, it indicates this benchmark is considered challenging.

## B. Full Environment Description

The RaidEnv platform offers extensive customization options for various game features, which can be controlled using different variables. Fig. 1 provides an overview of all the game components, while the detailed data type, range, and descriptions of the 17 character statistics and 17 game skills are described in the documentations.[3]

## C. Agent Parameters

Tables VIII and IX show the model hyperparameters and experimental settings used for training the play-testing (PT-MARL) and PCG agents (PCG-RL).

## REFERENCES

[1] E. R. Poromaa, "Crushing Candy Crush: Predicting human success rate in a mobile game using Monte-Carlo tree search," Master's Thesis, School Comput. Sci. Commun., KTH Roy. Inst. Technol., Stockholm, Sweden, 2017.

[2] S. F. Gudmundsson et al., "Human-like playtesting with deep learning," in *Proc. IEEE Conf. Comput. Intell. Games*, 2018, pp. 1–8.

[3] F. Lorenzo, S. Asadi, A. Karnsund, L. Cao, T. Wang, and A. H. Payberah, "Use all your skills, not only the most popular ones," in *Proc. IEEE Conf. Games*, 2020, pp. 682–685.

[4] T. Liu, Z. Zheng, H. Li, K. Bian, and L. Song, "Playing card-based RTS games with deep reinforcement learning," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 4540–4546.

[5] A. Summerville et al., "Procedural content generation via machine learning (PCGML)," *IEEE Trans. Games*, vol. 10, no. 3, pp. 257–270, Sep. 2018.

[6] L. Gisslén, A. Eakins, C. Gordillo, J. Bergdahl, and K. Tollmar, "Adversarial reinforcement learning for procedural content generation," in *Proc. IEEE Conf. Games*, 2021, pp. 1–8.

[7] M. Stephenson, E. Piette, D. J. Soemers, and C. Browne, "Ludii as a competition platform," in *Proc. IEEE Conf. Games*, 2019, pp. 1–8.

[8] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game AI: A multitrack framework for evaluating agents, games, and content generation algorithms," *IEEE Trans. Games*, vol. 11, no. 3, pp. 195–214, Sep. 2019.

[9] M. Wydmuch, M. Kempka, and W. Jaśkowski, "ViZDoom competitions: Playing doom from pixels," *IEEE Trans. Games*, vol. 11, no. 3, pp. 248–259, Sep. 2019.

[10] S. Karakovskiy and J. Togelius, "The mario AI benchmark and competitions," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 55–67, Mar. 2012.

[11] N. Bardet al., "The hanabi challenge: A new frontier for AI research," *Artif. Intell.*, vol. 280, 2020, Art. no. 103216.

[12] B. Cui, H. Hu, L. Pineda, and J. Foerster, "K-level reasoning for zero-shot coordination in Hanabi," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 8215–8228.

[13] C. Resnick et al., "Pommerman: A multi-agent playground," in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2018, pp. 1–6.

[14] K. Kurachet al., "Google research football: A novel reinforcement learning environment," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 4501–4510.

[15] H. Jia et al., "Fever basketball: A complex, flexible, and asynchronized sports game environment for multi-agent reinforcement learning," 2020, *arXiv:2012.03204*.

[16] M. Samvelyan et al., "The StarCraft multi-agent challenge," 2019, *arXiv:1902.04043*.

[17] B. Ellis et al., "SMACv2: An improved benchmark for cooperative multi-agent reinforcement learning," 2022, *arXiv:2212.07489*.

[18] J. Z. Leibo et al., "Scalable evaluation of multi-agent reinforcement learning with melting pot," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 6187–6199.

[19] J. Suarez, Y. Du, P. Isola, and I. Mordatch, "Neural MMO: A massively multiagent game environment for training and evaluating intelligent agents," 2019, *arXiv:1903.00784*.

[3][Online]. Available: https://github.com/CILAB-CT-GAME/RaidEnv/tree/main/docs

[20] M. Świechowski, T. Tajmajer, and A. Janusz, "Improving hearthstone AI by combining MCTS and supervised learning algorithms," in *Proc. IEEE Conf. Comput. Intell. Games*, 2018, pp. 1–8.

[21] P. Beau and S. Bakkes, "Automated game balancing of asymmetric video games," in *Proc. IEEE Conf. Comput. Intell. Games*, 2016, pp. 1–8.

[22] M. Carroll et al., "On the utility of learning about humans for human-AI coordination," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, Art. no. 465.

[23] M. Fontaine, Y.-C. Hsu, Y. Zhang, B. Tjanaka, and S. Nikolaidis, "On the importance of environments in human-robot coordination," in *Proc. Robot.: Sci. Syst.*, 2021, pp. 1–17.

[24] S. Reis, L. P. Reis, and N. Lau, "VGC AI competition–A new model of meta-game balance AI competition," in *Proc. IEEE Conf. Games*, 2021, pp. 01–08.

[25] S. Huang, S. Ontañón, C. Bamford, and L. Grela, "Gym-$\mu$RTS: Toward affordable full game real-time strategy games research with deep reinforcement learning," in *Proc. IEEE Conf. Games*, Copenhagen, Denmark, 2021, pp. 671–678.

[26] K. Sorochan and M. Guzdial, "Generating real-time strategy game units using search-based procedural content generation and monte carlo tree search," 2022, *arXiv:2212.03387*.

[27] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 172–186, Sep. 2011.

[28] R. Beal, T. J. Norman, and S. D. Ramchurn, "Artificial intelligence for team sports: A survey," *Knowl. Eng. Rev.*, vol. 34, 2019, Art. no. e28.

[29] J. Barambones, J. Cano-Benito, I. Sánchez-Rivero, R. Imbert, and F. Richoux, "Multi-agent systems on virtual games: A systematic mapping study," *IEEE Trans. Games*, vol. 15, no. 2, pp. 134–147, Jun. 2023.

[30] C. Stephens and C. Exton, "Measuring inflation within virtual economies using deep reinforcement learning," in *Proc. Int. Conf. Agents Artif. Intell.*, 2021, pp. 444–453.

[31] Y.-J. Gong et al., "Automated team assembly in mobile games: A data-driven evolutionary approach using a deep learning surrogate," *IEEE Trans. Games*, vol. 15, no. 1, pp. 67–80, Mar. 2023.

[32] M. Kaidan, T. Harada, C. Y. Chu, and R. Thawonmas, "Procedural generation of angry birds levels with adjustable difficulty," in *Proc. IEEE Congr. Evol. Comput.*, 2016, pp. 1311–1316.

[33] L. N. Ferreira and C. F. M. Toledo, "Tanager: A generator of feasible and engaging levels for angry birds," *IEEE Trans. Games*, vol. 10, no. 3, pp. 304–316, Sep. 2018.

[34] T. Shu, J. Liu, and G. N. Yannakakis, "Experience-driven PCG via reinforcement learning: A super Mario Bros study," in *Proc. IEEE Conf. Games*, 2021, pp. 1–9.

[35] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *Proc. Genet. Evol. Comput. Conf.*, 2018, pp. 221–228.

[36] Z. Jiang, S. Earle, M. Green, and J. Togelius, "Learning controllable 3D level generators," in *Proc. 17th Int. Conf. Found. Digit. Games*, 2022, pp. 1–9.

[37] I.-C. Baek, T.-G. Ha, T.-H. Park, and K.-J. Kim, "Toward cooperative level generation in multiplayer games: A user study in overcooked!," in *Proc. IEEE Conf. Games*, 2022, pp. 276–283.

[38] *Wowpedia*, "The World of Warcraft wiki encyclopedia." 2023. Accessed: Sep. 16, 2023. [Online]. Available: https://wowpedia.fandom.com/wiki/Wowpedia

[39] Lost Ark wiki. 2023. Accessed: Sep. 16, 2023. [Online]. Available: https://lostark.fandom.com/wiki/Lost_Ark_Wiki

[40] M. Seif El-Nasr et al., "Understanding and evaluating cooperative games," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2010, pp. 253–262.

[41] N. P. Zea, J. L. G. Sánchez, F. L. Gutiérrez, M. J. Cabrera, and P. Paderewski, "Design of educational multiplayer videogames: A vision from collaborative learning," *Adv. Eng. Softw.*, vol. 40, no. 12, pp. 1251–1260, 2009.

[42] A. Juliani et al., "Unity: A general platform for intelligent agents," 2020, *arXiv:1809.02627*.

[43] A. Cohen et al., "On the use and misuse of absorbing states in multi-agent reinforcement learning," 2021, *arXiv:2111.05992*.

[44] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. AAAI Conf. Artif. Intell.*, 2018, Art. no. 363.

[45] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.

[46] D. H. Wolpert and K. Tumer, "Optimal payoff functions for members of collectives," *Adv. Complex Syst.*, vol. 4, no. 02n03, pp. 265–279, 2001.

[47] J. T. Kristensen, A. Valdivia, and P. Burelli, "Estimating player completion rate in mobile puzzle games using reinforcement learning," in *Proc. IEEE Conf. Games*, 2020, pp. 636–639.

[48] K. R. McKee, J. Z. Leibo, C. Beattie, and R. Everett, "Quantifying the effects of environment and population diversity in multi-agent reinforcement learning," *Auton. Agents Multi-Agent Syst.*, vol. 36, no. 1, 2022, Art. no. 21.

[49] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, "PCGRL: Procedural content generation via reinforcement learning," in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2020, pp. 95–101.

[50] S. Earle, M. Edwards, A. Khalifa, P. Bontrager, and J. Togelius, "Learning controllable content generators," in *Proc. IEEE Conf. Games*, 2021, pp. 1–9.

[51] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[52] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics Intell. Lab. Syst.*, vol. 2, no. 1/3, pp. 37–52, 1987.

[53] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdiscipl. Rev.: Comput. Statist.*, vol. 2, no. 4, pp. 433–459, 2010.

[54] A. Abels, D. Roijers, T. Lenaerts, A. Nowé, and D. Steckelmacher, "Dynamic weights in multi-objective deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 11–20.