Dungeons & Replicants II: Automated Game Balancing Across Multiple Difficulty Dimensions via Deep Player Behavior Modeling

Johannes Pfau, Antonios Liapis, *Member, IEEE*, Georgios N. Yannakakis, *Senior Member, IEEE*, and Rainer Malaka

***Abstract*—Video game testing has become a major investment of time, labor, and expense in the game industry. Particularly the balancing of in-game units, characters, and classes can cause long-lasting issues that persist years after a game's launch. While approaches incorporating artificial intelligence have already shown successes in reducing manual effort and enhancing game development processes, most of these draw on heuristic, generalized, or optimal behavior routines, while actual low-level decisions from individual players and their resulting playing styles are rarely considered. In this article, we apply *deep player behavior modeling* to turn atomic actions of 213 players from six months of single-player instances within the MMORPG Aion into generative models that capture and reproduce particular playing strategies. In a subsequent simulation, the resulting generative agents ("replicants") were tested against common NPC opponent types of MMORPGs that iteratively increased in difficulty, respective to the primary factor that constitutes this enemy type (Melee, Ranged, Rogue, Buffer, Debuffer, Healer, Tank, or Group). As a result, imbalances between classes as well as strengths and weaknesses regarding particular combat challenges could be identified and regulated automatically.**

***Index Terms*—Artificial intelligence, automatic playtesting, game balance, massive multiplayer online games, user modeling.**

I. Introduction

THE exploding growth of the games industry has ramped up player demands for content and mechanics to extents that even large companies struggle to manage [1]. Beyond original content in new games, moreover, the online connectivity of game platforms and dedicated online games raise demands for changes, fixes, and innovations in already published titles. Notably, 80% of the 50 most popular games on the major distribution platform *Steam* require critical updates after launch [2]. Among the most important issues published game face is balancing, and prominent competitive online games that launched years ago still undergo persistent balance patches: examples include *StarCraft II* (Blizzard, 2010), *Dota 2* (Valve, 2013), and *Guild Wars 2* (NCSoft, 2012).

According to the definition of Sirlin [3], a game is *"balanced if a reasonably large number of options available to the player are viable"* (where viability sets the requirement of having many meaningful choices throughout a game), while *"players of equal skill should have an equal chance at winning."* Together with frequently desired asymmetrical configuration possibilities of these options, this inherently leads to combinatorial explosion, which can become hazardous for the enjoyability of the game and the satisfaction of its players. Even worse, Hullett *et al.* [4] highlight that balancing issues most of the time *"only become apparent after many months of play."* As opposed to straightforward fixable bugs, glitches, and solvability aspects, the trouble with balancing issues is that they do not only appear during the launch of a newly published game. Instead, balancing is an ongoing, repetitive task that is heavily influenced by the perceptions of the player community: *"after each patch, often the discussion begins again, factoring in new balancing or abilities for each class"* [5]. In the game industry, balancing is most often approached through long-term expert analysis, excessive human playtesting, and persistent debates with the community [6].

Due to these demands for playtesting before or after a game's launch, artificial intelligence (AI) has often been incorporated in the production and testing process [7]. For instance, AI can predict users' high-level behavior [8] (if they will leave the game, if they will make purchases) or their motivation levels [9] based on past trends in the userbase. Specifically with regards to *automated playtesting*, AI agents have been used to discover bugs or game crashes [10], constraint violations [11], [12] to identify dead-end game states [13] or unreachable states [14]. Notably, most of these agents follow *ad hoc* heuristics and constraints and do not necessarily match how players act in their games. The work of Holmgård *et al.* [15] explored the definition of multiple procedural personas which could test the game as archetypal players, although the definition of such personas still relied on designer input rather than direct human traces.

Manuscript received 19 March 2021; revised 8 August 2021 and 15 February 2022; accepted 4 March 2022. Date of publication 19 April 2022; date of current version 16 June 2023. The work of Johannes Pfau and Rainer Malaka was supported by the German Research Foundation (DFG) as part of Collaborative Research Center (SFB) under Grant 1320 EASE - Everyday Activity Science and Engineering, University of Bremen[1], subproject H2. The work of Antonios Liapis and Georgios N. Yannakakis was supported by the European Union's H2020 research and innovation programme under Grant 951911. (*Corresponding author: Johannes Pfau.*)

Johannes Pfau and Rainer Malaka are with the Digital Media Lab, University of Bremen, 28359 Bremen, Germany (e-mail: jopfau@ucsc.edu; malaka@tzi.de).

Antonios Liapis and Georgios N. Yannakakis are with the Institute of Digital Games, University of Malta, MSD 2080 Msida, Malta (e-mail: antonios.liapis@um.edu.mt; georgios.yannakakis@um.edu.mt).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TG.2022.3167728.

Digital Object Identifier 10.1109/TG.2022.3167728

[1][Online]. Available: http://www.easecrc.org/

This article applies *deep player behavior modeling* (DPBM) [16] which trains AI "replicants" in order to automatically test and balance a game. Within DPBM, individual decision-making from game states is mapped to a preference distribution of actions via machine learning, approximating the replication of individual players. In contrast to optimal or generalized models, the DPBM approach allows for the consideration of many (potentially viable) playing styles that players may employ instead of reducing it to a global decision-making module. In previous work, DPBM showed to be successful in generating agents capable of offering challenges on the same proficiency level [17] and convinced other players that they replicated individual behavior believably [18].

To test the efficacy of DPBM, in this work, we use a dataset of the popular massively multiplayer online role-playing game (MMORPG) *Aion* (NCSoft, 2008) consisting of atomic decision-making that was recorded throughout six months and 213 players in one-versus-one combat situations [17]. From this dataset, we generated DPBM-driven agents for all players and evaluated their proficiency against eight different types of enemy encounters in a 2-D benchmark that manipulated enemy-specific and general difficulty attributes of the enemy. For the empirical assessment of the resulting proficiencies, we used a metric that approximates the quality of performance in terms of effectiveness and efficiency.

We significantly extend previous work [19] that evaluated the balance between classes in one-versus-one combat scenarios against the environment or against other classes. This article focuses exclusively on player versus environment scenarios and explores how different players and classes perform against opponents that scale across multiple dimensions of difficulty. While previous work [19] observed the effects of increasing plain offensive (attack) and defensive (maximal hit points) attributes of the benchmark opponents, this work also interprets the effects of increasing various other difficulty dimensions (range, attack speed, self-improving buffs, target-weakening debuffs, healing, defense, and the number of enemies). The various enemies that replicants are tested against follow the patterns of the genre's main NPC enemy types (Melee, Ranged, Rogue, Buffer, Debuffer, Healer, Tank, or group of enemies, respectively). Evaluating the capabilities for automated game balancing and individual proficiency estimation, we aim to answer the following research questions.

1) *Can imbalances between in-game classes be detected through batched simulation analysis incorporating generative player modeling?*
2) *Does the segmentation across multiple difficulty dimensions aid in exposing the strengths and weaknesses of particular classes?*
3) *Can the field of automated game testing harness results of generative player modeling simulations to compute balanced configurations across classes?*

We hypothesize that agents that are representative of individual players' decision-making are able to detect differences in performance between classes and resemble the population closely. Under these conditions, DPBM should provide a viable technique to map behavioral patterns to proficiency scores and to inform automated game balancing empirically. Furthermore, we assume that situational strengths and weaknesses of particular classes are empirically detectable by contrasting various dimensions of difficulty. With insights from these evaluations, we expect to implement a methodology to automatically adjust in-game parameters toward interclass balance. This work contributes to games user research and game development in academia and industry by introducing a novel technique capable of enhancing game testing processes with the potential of reducing the associated effort.

II. Related Work

Automatic simulations of video game play have become viable and efficient alternatives and improvements to tedious and nonexhaustive human testing for the purpose of finding critical errors, solvability investigations, or parameter tuning. The majority of scientific approaches focus on detecting logical bugs or game crashes, such as Radomski *et al.* [11] or Varvaressos *et al.* [12] who identified violations of manually defined constraints via simulated play. Buhl *et al.* [20] highlighted the utility of autonomous testing routines in everyday continuous integration and continuous delivery pipelines by contrasting the amount of encountered bugs against previous developments without them. Zheng *et al.* [21] designed a game playing agent utilizing deep reinforcement learning, while Chan *et al.* [10] made use of a neuroevolution approach that on top of playing was able to report on the constellation and sequence of actions that lead to game malfunctions. Furthermore, Bécares *et al.* [22] mapped human tester playthrough records to semantic replay models using Petri nets, while Iftikhar *et al.* [23] and Schaefer *et al.* [24] introduced frameworks for autonomously testing generic games of the platformer or puzzle genre, respectively.

Several studies tackle solvability, such as those of Powley *et al.* [25] or Volkmar *et al.* [26] that assisted the level design of (procedurally generated) games by assuring potential solutions are feasible. Following a mixed-initiative [27] approach, Butler *et al.* [28] allowed the designer to specify the difficulty progression via a user interface while a constraint solver ensured the solvability of generated puzzles. Schatten *et al.* [14] simulated large-scale dynamic agent systems to test quest solvability in MMORPGs. Pfau *et al.* [13] introduced a generic adventure solver traversing point-and-click adventure games via reinforcement learning and reporting crashes, dead-ends, and performance issues. Van Kreveld *et al.* [29] and Southey *et al.* [30] assessed difficulty or interestingness approximations of levels or mechanics by machine learning of descriptive in-game metrics.

Regarding balancing, scientific approaches often build on simulations that iteratively assess balance criteria and dynamically tune in-game parameters based on the former. Jaffe *et al.* [31], García-Sanchez *et al.* [32], Volz *et al.* [33], Zook *et al.* [34], and De Mesentier Silva *et al.* [35] applied this paradigm to board or card games, which was amplified by Mahlmann *et al.* [36] by introducing procedurally generated cards on top of these simulations. In other genres, Beau and Bakkes [37] utilized Monte-Carlo Tree Search for balancing units of Tower

Defense games while Keehl and Smith [38] extended this to generic Unity games, Morosan and Poli [39] tweaked difficulty specifications in RTS and Arcade games after neuroevolution agents assessed these and Leigh *et al.* [40] dynamically balanced strategies though the coevolution of two competing agents playing a *Capture The Flag* game. Early work in AI and games research [41]–[43] introduced the notion of *interest* in prey-predator games and used metrics inspired by the challenge and curiosity factors of Malone [44] to tune the behavior of enemies in *Pac-Man*-like games so that the game maintains its interest levels for an individual player. A similar approach was taken for racing [45] and physical interactive games [46].

Similarly to the approach outlined in this work, Holmgård *et al.* [15] conflated atomic player behavior into procedural personas to simulate and test different play styles in a *Dungeon Crawler* game and Gudmundsson *et al.* [47] utilized atomic choices in order to predict the difficulty of various levels of a *Match-3*-Puzzle game. Nonetheless, even if some approaches process some kind of human player input, incorporating actual information about *individual* and *atomic* player behavior has not been tackled yet. Generative player modeling [48] connotes the generation of agent behavior from player-constructed models and has the potential to fuse automatic simulation methods with behavioral information, giving the developers the opportunity to receive practically immediate insights on which player strategies are popular, dominant and/or may require rework. Further generative player modeling is able to inform developers on how parameter tuning will likely alter the outcome of strategies before presenting it to the community, how implemented dynamic difficulty approaches can be informed about parameter thresholds, and how to automatically balance game mechanics after large-scale permutations of classes, setups, parameters, and behavior in all stages of development.

III. Deep Player Behavior Modeling in Aion

This section describes the selected game, training dataset, and player modeling methodology taken to create the replicants used in experiments of this article.

A. Game Environment

The MMORPG *Aion* (NCSoft, 2008) was chosen as a representative game within a genre that considerably suffers from the aforementioned balancing issues. *Aion* includes a number of character classes which encompass a typical set of playstyles. *Melee* classes (Gladiator, Templar, Assassin) mainly deal close-combat damage, in contrast to *Magic* classes (Sorcerer, Spiritmaster, Gunner) or Rangers. *Heal* classes (Cleric, Bard) deal less damage but offer additional support, while Chanters excel at the latter. Even if many in-game situations involve multiplayer constellations, all classes are able to perform on their own in principle. Combat is mainly fought out by activating skill actions that harm the opponent(s) and/or benefit the player character. Different strategies are possible depending on the skills used, and their sequence. While these strategies rarely maximize efficiency, they resemble situational preferences that emerge in personal play styles, such as improving one's offensive or defensive capabilities or controlling the opponent's actions.

TABLE I
Archetypes, Classes, Unique Skills Per Class, and Number of Players of Each Class in the Dataset

Archetype	Class	# Skills	# Players
Melee	Gladiator	78	33
	Templar	56	19
	Assassin	57	17
Magic	Sorcerer	52	20
	Spiritmaster	51	18
	Gunner	42	10
Ranged	Ranger	53	13
Support	Chanter	57	19
Heal	Cleric	48	25
	Bard	78	39

B. Dataset of Aion Players

In order to train the models with the necessary low-level state-action mapping, primary player data were collected among experienced *Aion* players [49]. Over the course of six months, 213 players were recorded within a daily single-player dungeon instance in challenging one-versus-one combat situations [17], totaling to $\sim 280\,000$ actions. Table I shows the number of players per class in the dataset. For the best compromise between ecological validity of the data and modifiability and operationality of the game code, player recordings and later benchmark simulations took place on a private server of *Aion*. The data were collected between July and December 2019. The dataset, explanations and precomputed examples have been published[2] to the Open Science Framework (OSF).

C. Deep Player Behavior Modeling

DPBM realizes individual generative player modeling by assessing atomic player behavior in a state-action architecture and establishes a mapping among these via machine learning [16]. For generating a replicative agent that is representative of a single individual, the recorded behavioral data from all relevant observations was retrieved from the underlying database and fed into a feed-forward multilayer perceptron (MLP) employing a softmax activation function, a stochastic gradient descent optimizer with categorical crossentropy loss function and trained via backpropagation. The input layer consists of 22 nodes describing the current game state plus a set of nodes representing the preceding skill. Consisting of the same set of skill nodes, the output layer characterizes the probability distribution of action choices with respect to the individual player and the input situation (cf., Fig. 1). The number of skill nodes varied per class, $(42 - 78)$, as shown in Table I.

The network was initialized randomly, contained four hidden layers with equal size to the input layer and was trained over 1000 epochs, based on insights from previous work [16]–[18], [50], [51]; benchmarks prior to the study also indicated diminishing returns with more parameters (more and wider layers) and epochs.

[2][Online]. Available: https://osf.io/3ktc6/

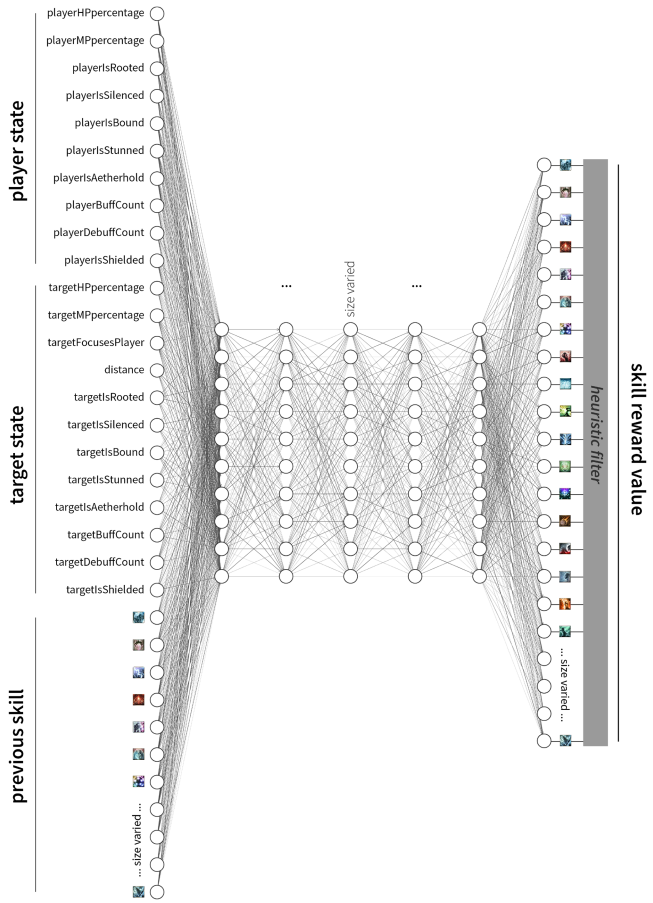

Fig. 1. Neural network architecture for DPBM, with game state as input and skill usage preference as output. All layer sizes varied depending on the skill set of the player class.

When exposed to the testing environment, the trained model was applied generatively to retrieve a set of action probabilities given the occurring state description at real-time. After a weighted choice, the resulting skill was executed, followed by querying the DPBM for the next situation, effectively approximating the learned behavior from the original player's battles. Based on the player modeling taxonomy of Yannakakis *et al.* [7], [52], this implementation realizes a *model-free* (bottom–up) player modeling approach mapping *gameplay data* to actions via *classification*. According to the player modeling description framework of Smith *et al.* [48], DPBM directly utilizes *game actions* (domain) to *generate* (purpose) *individually* (scope) modeled behavior by means of *induced* (source) training of machine learning techniques.

This article uses DPBM replicants pretrained from previous work [17]. Each replicant is trained on the data of a single user. Overall, testing prediction accuracies of the employed DPBMs averaged to 61.3% within Top-1, 75.3% within Top-5, and 81.3% within the Top-10 most probable actions (using a 80–20 holdout validation method). Similar testing accuracies have led to convincing gameplaying agents in prior work, which tested whether players would notice when real players were replaced in live online matches by their DPBM substitute [18] and to agents that produced player statements about "the ability to learn from previous battles and the adaptation to the player's own behavior, combos, rotations and/or strategies to their enemy"[17].

TABLE II
PvE Enemy Types and Their Difficulty Parameters

PvE enemy	Difficulty parameters	
	Primary	Secondary
Melee	Damage	Hit Points
Ranged	Range	
Rogue	Attack speed	
Buffer	Buffs	
Debuffer	Debuffs	
Healer	Heal amount	
Tank	Defense	
Many	Numbers	

D. Proficiency Metric

To assess how agents fare in different combat situations, we construct a proficiency metric which considers four variables measured at the end of a one-versus-one combat situation.

1) The binary value of having won against the opponent (w).
2) The normalized temporal duration of the fight (t).
3) The agent's remaining hit point (HP) percentage (hp_a).
4) The opponent's remaining HP percentage (hp_o).

All variables lie between 0 and 1, and the final proficiency score ϕ consists of a weighted sum of these measures normalized over weights and the sum of observations (n). Equation (1) shows how the final proficiency score is calculated, with 0 being a worst-case scenario and 1 being a best-case scenario. Note that all weights ($\alpha, \beta, \gamma, \delta$) are 1 in this article, offering the same importance to each of the four variables

$$\phi = \sum_{i,j=1}^{n} \frac{\alpha w + \beta(1-t) + \gamma hp_a + \delta(1-hp_o)}{(\alpha+\beta+\gamma+\delta)n^2}. \quad (1)$$

IV. Experimental Setup

This article focuses on the proficiency of different players and classes against different scripted encounters with enemies in a player versus environment (PvE) setting. Using the 213 DPBM-driven agents as replicants of the players that participated in the data collection experiment, eight scenarios are designed with different types of encounters typical of PvE challenges in Aion. Each replicant fights against different versions of the same encounter per condition, where two difficulty parameters are adjusted incrementally. The primary difficulty parameter is idiosyncratic to this type of enemy (e.g., more difficult ranged enemies may have more range) while the secondary difficulty parameter is always the number of hit points (HP) that each enemy has. Table II shows the eight different conditions, and Table III explains each parameter and how it scales. Each replicant fights against 100 versions of the encounter per condition (cf., Fig. 2), at different values for each difficulty parameter: each parameter is scaled iteratively for ten total iterations per parameter. Note that the *Many* condition only scales the number of enemies five times instead of ten since the encounters become extremely difficult when fighting more than five enemies. For the *Many* condition, therefore, 50 different parameter combinations are

TABLE III
DESCRIPTION OF THE DIFFICULTY PARAMETERS, AND HOW THEY SCALE IN EVERY ITERATION

Param.	Explanation	Scaling
Hit Points	Starting and maximum hit points (HP) of each enemy	25% increase
Damage	Damage applied per attack	25% increase
Range	Maximum distance for attack, and spawn offset	25% increase
Attack speed	Frequency of attacks	25% increase
Buffs	Cumulative strengthening spells applied to self	Additional buff
Debuffs	Cumulative weakening spells applied to opponent	Additional debuff
Heal amount	HP healed at regular interval, up to the maximum HP	25% increase
Defense	Reduction of physical and magical damage done by opponent	25% increase
Numbers	Number of (identical) enemies in the encounter	+1 enemy per 2 iterations

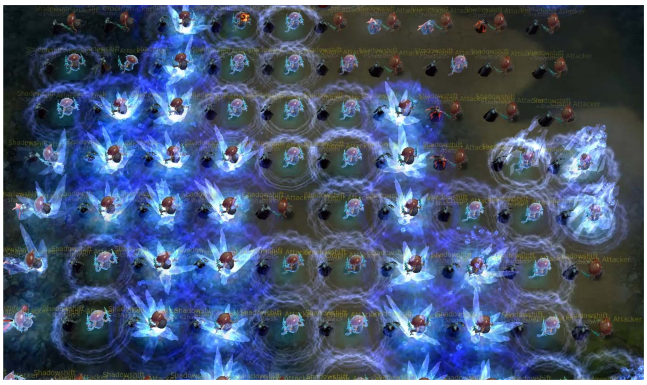

Fig. 2. In-game screenshot of the PvE benchmark in *Aion*, with a replicant fighting in 100 combat encounters against enemies with different difficulty parameters.

tested rather than 100 since the replicant fought two encounters for each parameter combination.

This article focuses on the proficiency metric (ϕ) presented in Section III-D and explores how it fluctuates depending on the player, the player's class, the type of enemy (condition), and the values of the different difficulty parameters. When calculating ϕ for the Many condition, the hit points [hp_o in (1)] is the average remaining HP of all opponents. When comparing ϕ scores directly, we apply the Welch's t-test for ascertaining significant differences between population means. When assessing the dependence between ϕ scores of different conditions, or ϕ scores with difficulty parameters, we instead apply the Kendall's rank correlation coefficient (τ), which measures the ordinal association between two rankings [53]. Kendall's τ is 1 if the agreement between the two rankings is perfect, and -1 if one ranking is the reverse of the other. In all reported results, the threshold for significance is set at $\alpha = 0.05$.

V. RESULTS

This section explores how replicants' proficiency scores in 800 encounters against PvE enemies fluctuate depending on the player, the player's class, the type of enemy (condition), and the values of the different difficulty parameters.

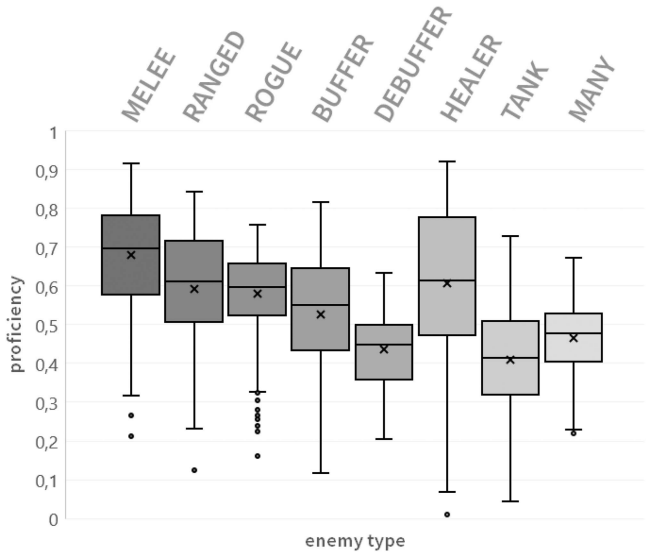

Fig. 3. Resulting ϕ proficiency of all 213 replicants against the eight different PvE encounter conditions.

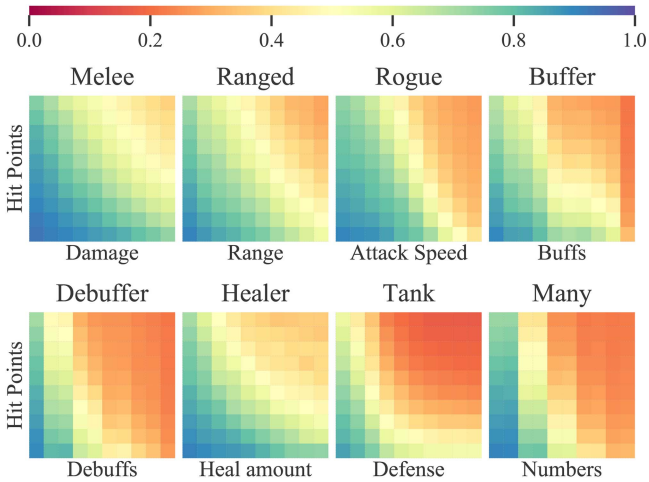

Fig. 4. Heatmap of ϕ proficiency scores for each different encounter across primary and secondary difficulty parameter setups, averaged over all replicants.

A. General Findings

Fig. 3 shows the average proficiency score of all 213 replicants trained in this study against PvE encounters of different conditions. It is evident that different conditions were more challenging than others, with the easiest being the Melee encounter condition ($\bar{\phi} = 0.68$) and the hardest being the Tank condition ($\bar{\phi} = 0.41$). Due to these differences, Section V-B applies z-normalization to the proficiency scores per condition based on the distribution of all ϕ scores of all replicants in the same condition.

Fig. 4 shows heatmaps of the average proficiency score of all 213 players in each difficulty parameter combination per condition. Unsurprisingly, iteratively scaling any of the two difficulty parameters lowers the replicants' proficiency in the

TABLE IV
KENDALL τ CORRELATION RESULTS BETWEEN PROFICIENCY SCORE AND DIFFICULTY PARAMETERS, CONSISTENTLY SIGNIFICANT

	Damage	Range	Att. speed	Buffs	Debuffs	Heal amount	Defense	Numbers	Hit points
Overall	−0.31	−0.28	−0.27	−0.28	−0.20	−0.32	−0.31	−0.26	−0.38
Gladiator	−0.41	−0.40	−0.36	−0.40	−0.25	−0.32	−0.37	−0.35	−0.42
Templar	−0.45	−0.44	−0.35	−0.42	−0.24	−0.43	−0.38	−0.29	−0.49
Assassin	−0.34	−0.38	−0.29	−0.37	−0.21	−0.41	−0.28	−0.28	−0.43
Ranger	−0.41	−0.33	−0.32	−0.32	−0.25	−0.36	−0.31	−0.30	−0.42
Sorcerer	−0.37	−0.29	−0.26	−0.28	−0.18	−0.36	−0.33	−0.28	−0.33
Spirit-master	−0.25	−0.22	−0.20	−0.26	−0.20	−0.38	−0.39	−0.22	−0.38
Cleric	−0.30	−0.27	−0.21	−0.24	−0.20	−0.33	−0.26	−0.23	−0.36
Chanter	−0.39	−0.57	−0.34	−0.46	−0.26	−0.36	−0.36	−0.38	−0.34
Gunner	−0.31	−0.33	−0.27	−0.31	−0.23	−0.37	−0.36	−0.23	−0.34
Bard	−0.29	−0.27	−0.26	−0.27	−0.22	−0.36	−0.38	−0.24	−0.38

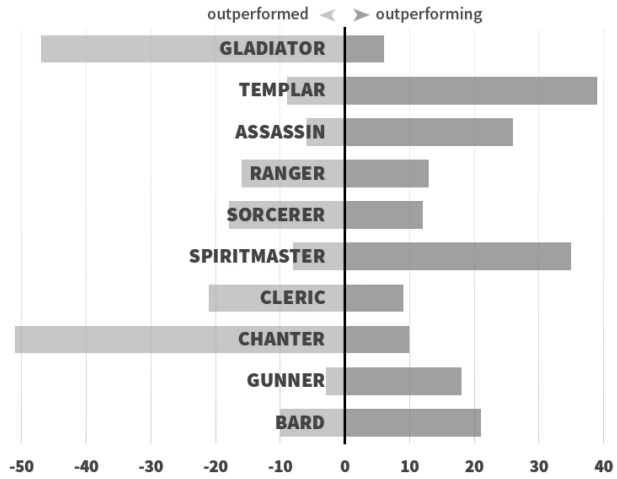

Fig. 5. Boxplot of normalized ϕ proficiencies of classes throughout the eight different encounters.

encounter. However, it is interesting to note that different difficulty parameters affect the proficiency differently. For instance, while Buffer and Debuffer encounters with low HP are fairly manageable by replicants, at the highest value of buff or debuff parameters the proficiency falloff is substantial, even with very low-HP encounters. One can assume that a Debuffer enemy weakens their opponent so much that none of the replicants' abilities deal any damage in this encounter. In contrast, at low HP the Tank opponent is manageable even with high values in defense. This means that the replicant's abilities deal enough damage (despite resistances) to defeat the low-HP opponent. This incongruity may be an artifact of the value ranges chosen for these two conditions, but may also point to a difference in the way the game engine handles debuffs and damage resistance which could be helpful for game designers.

To assess the impact of different difficulty parameters on the replicants' proficiency, Table IV shows the correlation of each parameter with the overall ϕ scores (of all 213 replicants), as well as per class; we discuss the latter in the next section. As intended, there is a significant negative correlation between every difficulty parameter and replicants' proficiency. Interestingly, the more consistent impact is from the damage and defense parameters, and the most "erratic" way of controlling difficulty was the Debuffs parameter. We investigate this further below.

Finally, it is interesting to evaluate whether the replicants themselves were consistent in their proficiency across different conditions. Since each replicant was trained on a specific player, the hypothesis is that a player would perform better or worse than other players against all encounters regardless of enemy type. We should note that the class information is also embedded in this evaluation, as a good player playing an "inferior" class would likely perform worse than a good player playing a "superior" class. To assess this, the average performance of one replicant across all 100 fights in one condition is ranked and compared with replicants' ranking in each other condition. As expected, the rankings of individual replicants largely match across conditions, with Kendall τ values ranging from 0.17 to 0.74 (with an average of 0.43 across all 28 comparisons). The most divergent condition was the Tank: the lowest Kendall τ scores overall are between Tank and three other conditions: Melee ($\tau = 0.17$), Debuffer ($\tau = 0.21$), and Healer ($\tau = 0.24$).

B. Differences Between Classes

To approach the actual balance between classes in these PvE encounters, the proficiency scores of replicas of each class were compared against replicas of other classes. After the calculation of one-way ANOVAs over these conditions, significant differences between classes remained for each encounter ($p < 0.05$ in all cases). Based on pairwise Welch's t-tests between scores of different classes in the same condition, the number of times one class performed significantly better or significantly worse than another class is summarized in Fig. 6. The chart shows that the Gladiator and Chanter classes generally performed worse than other classes across conditions, while the Templar, Spiritmaster, and Assassin performed overall better than most. We discuss which conditions each class performed better in below.

Fig. 5 shows the proficiencies of each class in different conditions: note that for each condition the values of each condition are z-normalized based on trends across all replicants' performance for that condition. Comparing these normalized scores within each class via one-way ANOVAs, we identify no significant differences across conditions for Rangers, Sorcerers, Clerics, and Gunners, but there are significant differences across conditions for all other classes. Further Welch's t-tests identified the particular deviant cases, resulting in the following insights, compared to other conditions.

1) Gladiators performed significantly worse against Debuffers.
2) Templars performed significantly worse against Melee and Healers, but significantly better against Tanks.
3) Assassins performed significantly worse against Melee and Tanks, but significantly better against Ranged.
4) Spiritmasters performed significantly worse against Tanks.
5) Chanters performed significantly better against Rogues and Tanks.

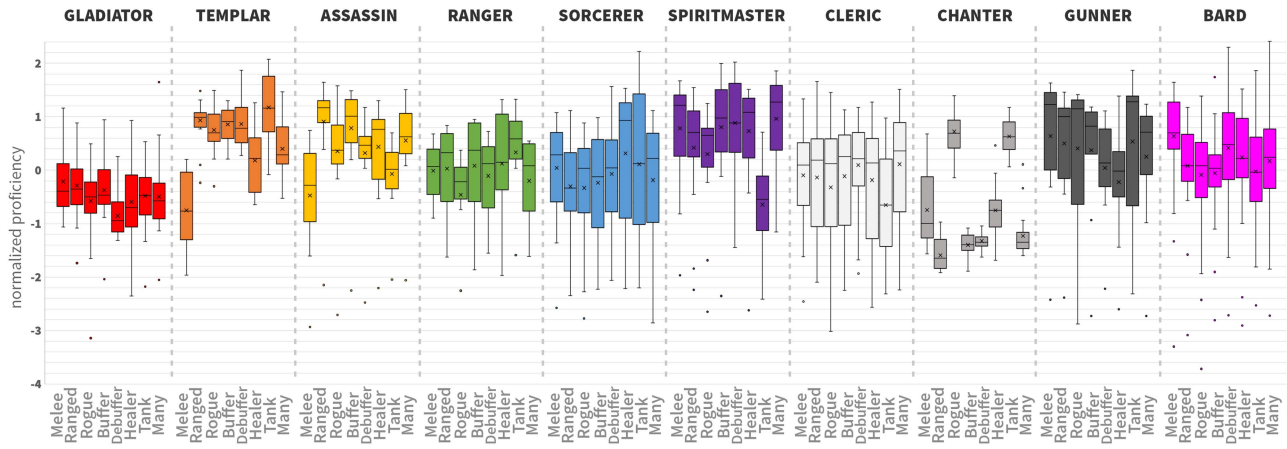

Fig. 6. Classes significantly outperforming (or being outperformed by) other classes, accumulated over all eight conditions.

6) Bards performed significantly better against Melee and Debuffers.

It is clear that certain classes are better equipped at handling high-defense targets (Templar, Chanter) than others (Spiritmaster, Assassin) as the former come with substantially better survivability while keeping sufficient sustained damage output. Interestingly, all melee classes (Gladiator, Templar, Assassin) perform poorly against Melee enemies compared to classes dealing ranged magic damage (Spiritmaster, Gunner, and Bard). Meanwhile, the Cleric, Ranger, Sorcerer, and Gladiator classes are most consistent in their performance across conditions (based on low standard deviation between the z normalized proficiency scores). The Gladiator performs consistently poorly, while the other classes are fairly average in their performance (i.e., close to the proficiency score of all replicants in that condition).

C. Balancing Encounters for Different Classes

The vast corpus of individual replicants' combat outcomes against a large and diverse set of encounters allows for the regulation of such outcomes. Automated balancing can follow various methodologies, e.g., adjusting passive class attributes or skill parameters, or tuning the difficulty dimensions of the encounters. While the latter is only restricted to solo play where no other classes or players are able to interfere in the regulation, the former can influence interclass balance and undesirably impact player versus player situations. Thus, this work approaches automated balancing by altering the encounters for each respective class, which can additionally be transferred to the more specific level of individual players, if aiming for dynamic difficulty adjustment [54]. Using the proficiency scores for every encounter condition, difficulty dimension, and all replicas (i.e., 170 400 combat situations), a mean proficiency score across the board is derived which we define as our target proficiency ($\phi_t = 0.541$). We identify which difficulty parameter combination per class and condition (i.e., each cell in the heatmaps of Fig. 4) has the closest proficiency to the target (as an absolute difference) for each in-game class. This results in ten ideal enemies of each of the eight conditions. Fig. 7 shows each ideal enemy and their distribution with respect to individual player proficiencies via a kernel density estimate (KDE) plot. The KDE plot can visualize which areas of the parameter space are better suited for the majority of replicants of each class.

When comparing proficiency scores between classes following these regulation targets, no significant differences remain (cf., Fig. 8), according to one-way ANOVAs for each condition ($p > 0.05$). There are interesting differences between the KDE plots across classes, but also when considering the average proficiency heatmaps of Fig. 4. As expected, melee classes (Gladiator, Templar, Assassin) prefer Melee enemies with low damage, while Bards can handle enemies with high Melee damage or long range. The KDE plots provide more feedback to designers regarding the ideal ranges for each class and enemy type, and can be updated when additional player data becomes available and additional replicants are trained.

It should be noted that the optimal encounter could be calculated as above based on any arbitrary target proficiency score, and the ideal enemy parameter pairing per replicant can be used to personalize the encounter to each individual player rather than, e.g., via class-wide adjustments.

VI. Discussion

An important dimension of inquiry in this article was the imbalance between different character classes. ANOVAs and subsequent posthoc tests revealed significant differences in proficiency between player replicants of different classes. Accumulated over all conditions, these might indicate imbalances of the classes, yet it should be interpreted with respect to the underlying design guidelines. For instance, the relatively low proficiency scores of the Chanter class likely stems from their reliance on other players, as they constitute the game's main support class. Still, under the assumption that primarily damage-dealing classes should be equally viable, certain discrepancies emerge that point to certain classes (such as Templar, Spiritmaster, or Assassin) outperforming most of the other classes in many situations, while others (such as Gladiator and Chanter) are

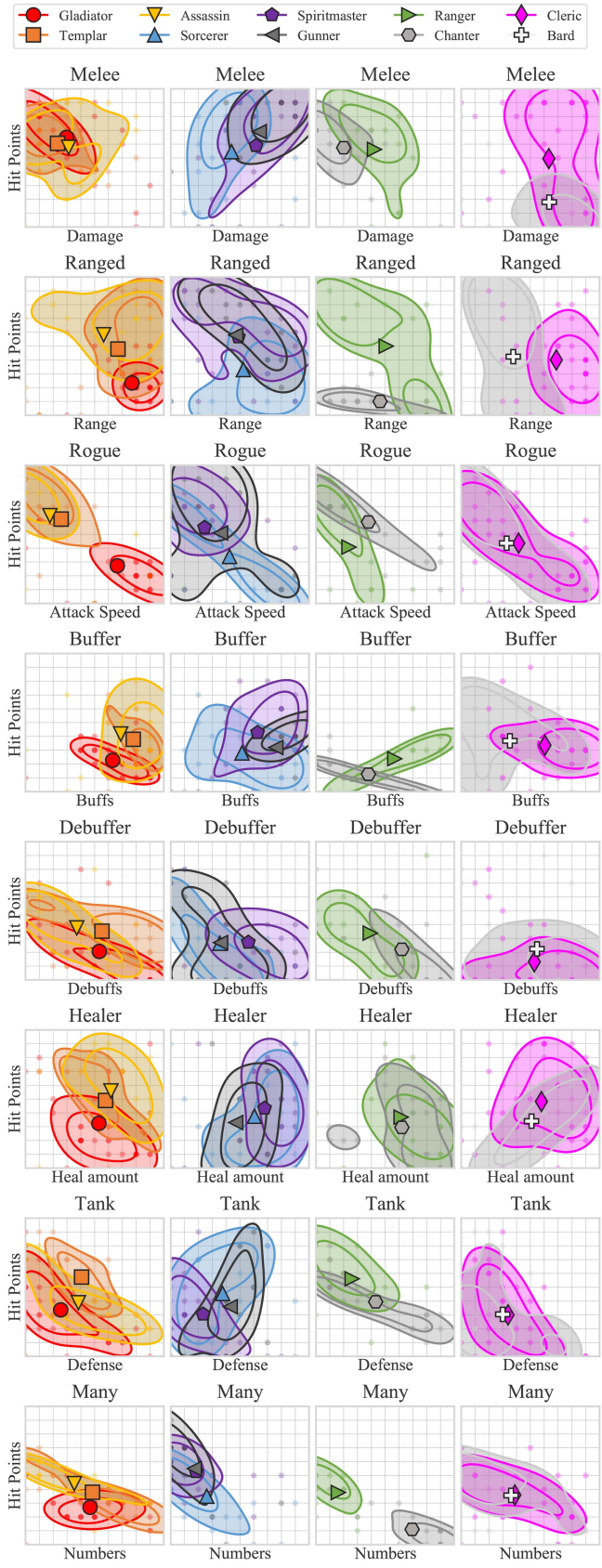

Fig. 7. Ideal enemies (markers) that lead to a balanced configuration among classes. Distributions of individual player performances within 10% of the target ϕ_t are also shown.

mostly outperformed by any other class. On the other hand, some classes (such as Rangers and Sorcerers) appear rather balanced. As the overall difficulty outcomes of the various conditions turned out to be significantly different, only normalizing the proficiency scores could shed light on the strengths and weaknesses within particular classes. Most of the time, proficiencies stayed similar for the respective class when comparing their performances across the encounters. This is underlined by the significant positive correlations between score ranks among conditions—both for individual replicants as well as for class ranks—indicating that well-performing replicants/classes also performed well in other conditions. Yet, several significant deviations highlighted encounters that are particularly difficult for certain classes; this might stem from the underlying design intentions. For instance, Spiritmasters performed poorly against Tank enemies, since their main damage contribution comes from damaging debuffs that these enemies are more likely to resist; Templars struggle mainly when the enemy's sheer attack power is overwhelming, but can survive most other conditions while applying medium damage; or Assassins that excel against ranged enemies, as they can quickly overcome the distance between them.

It should be noted that balance does not necessary imply that every class should be equally proficient in every in-game situation. When taking all of the possible encounters that a game offers into account, however, different classes should be equally viable. No class should be outperformed or outperforming others consistently, and weaknesses of particular classes against certain encounters should be compensated with advantages in other situations. Imbalances with respect to this assumption could either be diminished by dynamically adjusting difficulty parameters of PvE enemies depending on the estimated proficiency of the player class or by directly regulating class-specific skills/attributes. While the former method is restricted to single-player situations (as proficiencies of multiple players are not trivially merged and inter-player dynamics are not captured within this benchmark), the latter could end up distorting the inter-class balance in player versus player situations. If aiming for balanced proficiency of all classes against multiple difficulty conditions and each other, the former approach has shown to compute optimal difficulty parameter constellations that eventually lead to balanced performances between classes throughout all conditions. As the utilized notion of target proficiency is flexible, the internal difficulty can be manually scaled by designers while still maintaining a balanced state among classes. Moreover, it is not restricted to regulate balance between entire classes, but can additionally be used to compute parameter constellations for clusters of players or individual players, effectively implementing dynamic difficulty adjustment. Based on the empirical evidence presented, our previously posed research questions can be answered as follows.

1) *Imbalances between in-game classes can be detected by generative player modeling and iterative simulations.*
2) *Assessing multiple difficulty dimensions can reveal class-specific strengths and weaknesses and offer overall insights.*

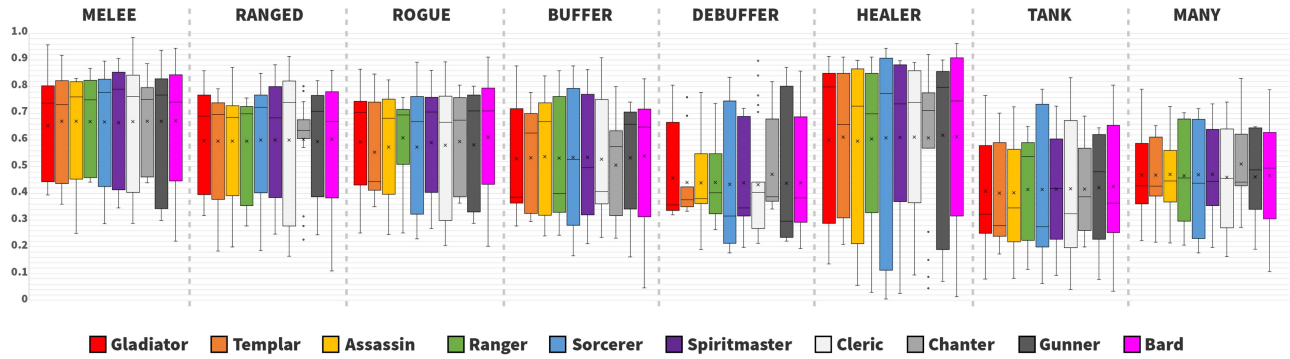

Fig. 8. Boxplot of ϕ proficiencies of classes when exposed to their ideal enemy (for all eight different encounter types). Within each condition, no significant balance difference remains.

3) *Using atomic replicants of a whole player population, parameter constellations can be computed that eventually lead to a balanced state between classes across several in-game difficulty dimensions.*

VII. Limitations and Future Work

During the implementation of this approach, different constraints and assumptions had to be taken into account that eventually lead to a number of limitations. Perhaps most importantly, classes (especially in MMORPGs) are often designed to vary in versatility within different situations or against different classes. This includes classes that benefit greatly from party-play versus classes that are tailored for single-player situations, those that focus on dealing damage to many enemies instead of single targets or not primarily focused on dealing damage (being busy with tanking, healing, or supporting otherwise). Nevertheless, the presented technique is not constrained to damage dealing, but overall one-on-one versatility. DPBM can quantify these differences to inform game developers whether their intended design aligns with the actual outcomes of a population playing it. Evidently, this requires data from a player population to employ the testing procedures, which limits its versatility before the game's launch. However, it is applicable for early access titles or prelaunch scenarios, never-ending balance observations (and predictions), and for benchmarking novel challenges introduced with later patches or DLCs, given that abundant player data can be collected, a sufficient telemetry infrastructure is realizable and the developers have the means to record individual players over longer periods of time. Apart from individually learning replicants, optimally playing agents (e.g., by self-training/reinforcement learning) could be used as a baseline to test whether DPBM indeed approximates the real population better. On another note, we normalized equipment and other relevant configurations throughout all characters in order to filter out the influence of different attribute stats. A closer (yet very temporary) approximation of the overall population capability could be realized with this approach if the equipment range was taken into consideration. Notably, replicants were only trained on data stemming from battles against their own class [17], and thus, did not adjust their strategies against different encounter types. This likely distorted the results and should be repeated when enough data of the respective situations are given; however, it does not diminish the potential of DPBM.

For future work, we primarily seek to refine behavior modeling by introducing more variables, such as global movement information (encompassing higher level goals) or the estimation of individual players' precision and their temporal cognitive computation demand. Apart from the inclusion of the nine difficulty parameters, the challenge of the opponent encounters can further be examined by altering the skill sets, decision making or movement behavior of enemies. The simulations themselves can likely be sped up by calculating battles without graphical representations. Instead of altering opponent parameters for balancing, we additionally want to explore the automated adjustment of in-game classes with respect to player versus environment, as well as player versus player settings. For this, an iterative procedure of attunement and resimulation would be expedient, in that the largest proficiency mismatch between classes is detected, adjusted in favor of the inferior class and affected matchups are resimulated, in multiple iterations up to a predefined threshold.

Finally, the applicability of this approach will be investigated with respect to significantly more complex multiplayer situations, such as in adjusting boss battles for a population or simulating large-scale competitive sieges between replicants, throughout multiple player experience evaluations. Furthermore, if a mapping from mere behavioral patterns to in-game proficiency can be constructed—e.g., via machine learning—this prediction might augment matchmaking, bringing together players with approximate skill levels more accurately, for both competitive as well as cooperative play.

VIII. Conclusion

This article explored how AI agents which replicate the action-by-action decision-making of individual players could be applied to test the impact of enemy types and their parameters in combat encounters against different players and different character classes. The article made use of player traces collected from the MMORPG *Aion* and drew conclusion regarding the character classes and enemy types of this game through an extensive set of combat simulations. The proficiency score which aggregates several properties of the combat outcome and the 2-D

exploration of general and enemy-specific parameters allows for a straightforward visualization of the performance of each player, each class, or the entire userbase in a way that would be easily understood by game designers. Moreover, it is very straightforward to identify one or many optimal encounters for each enemy type based on the distance to a target proficiency score. Eventually, this allows regulation either on a per-class basis or via personalized difficulty adjustment, tailored to individual players. All steps, from model computation over benchmark simulations up to the final balancing regulation, can be executed in a fully autonomous way (requiring only playtraces of the player population), realizing automated game balancing. The process can also be supervised by designers, providing parameter spaces, visualizations, and reports on classes' strength and weaknesses which can inform the game's development. Even though this functionality could only be shown for a single game so far, we claim that this procedure also holds for other games and genres, as long as one provides entities to balance (e.g., classes), meaningful benchmark simulations (e.g., combat situations) and representative low-level interaction data of a player population.

REFERENCES

[1] M. Washburn Jr, P. Sathiyanarayanan, M. Nagappan, T. Zimmermann, and C. Bird, "What went right and what went wrong: An analysis of 155 postmortems from game development," in *Proc. Companion Proc. Int. Conf. Softw. Eng.*, 2016, pp. 280–289.

[2] D. Lin, C.-P. Bezemer, and A. E. Hassan, "Studying the urgent updates of popular games on the steam platform," *Empirical Softw. Eng.*, vol. 22, no. 4, pp. 2095–2126, 2017.

[3] D. Sirlin, "Balancing multiplayer competitive games," Dec. 2001. [Online]. Available: https://www.sirlin.net/articles/balancing-multiplayer-games-part-1-definitions

[4] K. Hullett, N. Nagappan, E. Schuh, and J. Hopson, "Empirical analysis of user data in game software development," in *Proc. Int. Symp. Empirical Softw. Eng. Meas.*, 2012, pp. 89–98.

[5] C. Lewis and N. Wardrip-Fruin, "Mining game statistics from web services: A world of warcraft armory case study," in *Proc. Found. Digit. Games Conf.*, 2010, pp. 100–107.

[6] J. Pfau, J. D. Smeddinck, and R. Malaka, "The case for usable AI: What industry professionals make of academic AI in video games," in *Proc. Extended Abstr. Annu. Symp. Comput.-Hum. Interact. Play*, 2020, pp. 330–334.

[7] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Berlin, Germany: Springer-Verlag, 2018.

[8] J. Runge, P. Gao, F. Garcin, and B. Faltings, "Churn prediction for high-value players in casual social games," in *Proc. Comput. Intell. Games Conf.*, 2014.

[9] D. Melhart, A. Azadvar, A. Canossa, A. Liapis, and G. N. Yannakakis, "Your gameplay says it all: Modelling motivation in Tom Clancy's The Division," *IEEE Conf. Games (CoG)*, pp. 1–8, 2019, doi: 10.1109/CIG.2019.8848123.

[10] B. Chan, J. Denzinger, D. Gates, K. Loose, and J. Buchanan, "Evolutionary behavior testing of commercial computer games," in *Proc. Evol. Comput. Congr.*, 2004, pp. 1:125–1:132.

[11] S. Radomski and T. Neubacher, "Formal verification of selected game-logic specifications," in *Proc. EICS Workshop Eng. Interactive Comput. Syst. SCXML*, 2015, pp. 30–34.

[12] S. Varvaressos, K. Lavoie, S. Gaboury, and S. Hallé, "Automated bug finding in video games: A case study for runtime monitoring," *Comput. Entertainment*, vol. 15, no. 1, 2017.

[13] J. Pfau, J. D. Smeddinck, and R. Malaka, "Automated game testing with ICARUS: Intelligent completion of adventure riddles via unsupervised solving," in *Proc. Extended Abstr. Annu. Symp. Comput.- Hum. Interact. Play*, 2017, pp. 153–164.

[14] M. Schatten, B. O. Đurić, I. Tomičič, and N. Ivkovič, "Automated MMORPG testing-an agent-based approach," in *Proc. Int. Conf. Practical Appl. Agents Multi-Agent Syst.*, 2017, pp. 359–363.

[15] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius, "Automated playtesting with procedural personas through MCTS with evolved heuristics," *IEEE Trans. Games*, vol. 11, no. 4, pp. 352–362, Dec. 2019.

[16] J. Pfau, J. D. Smeddinck, and R. Malaka, "Towards deep player behavior models in MMORPGs," in *Proc. Annu. Symp. Comput.- Hum. Interact. Play*, 2018, pp. 381–392.

[17] J. Pfau, J. D. Smeddinck, and R. Malaka, "Enemy within: Long-term motivation effects of deep player behavior models for dynamic difficulty adjustment," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2020, pp. 1–10.

[18] J. Pfau, J. D. Smeddinck, I. Bikas, and R. Malaka, "Bot or not? User perceptions of player substitution with deep player behavior models," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2020, pp. 1–10.

[19] J. Pfau, A. Liapis, G. Volkmar, G. Yannakakis, and R. Malaka, "Dungeons & replicants: Automated game balancing via deep player behavior modeling," in *Proc. Conf. Games.*, 2020, pp. 431–438.

[20] C. Buhl and F. Gareeboo, "Automated testing: A key factor for success in video game development. case study and lessons learned," in *Proc. Pacific NW Softw. Qual. Conf.*, 2012, pp. 1–15.

[21] Y. Zheng *et al.*, "Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning," in *Proc. Int. Conf. Automated Softw. Eng.*, 2019, pp. 772–784.

[22] J. H. Bécares, L. C. Valero, and P. P. G. Martín, "An approach to automated videogame beta testing," *Entertainment Comput.*, vol. 18, pp. 79–92, 2017.

[23] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, "An automated model based testing approach for platform games," in *Proc. Int. Conf. Model Driven Eng. Lang. Syst.*, 2015, pp. 426–435.

[24] C. Schaefer, H. Do, and B. M. Slator, "Crushinator: A framework towards game-independent testing," in *Proc. Int. Conf. Automated Softw. Eng.*, 2013, pp. 726–729.

[25] E. J. Powley, S. Colton, S. Gaudl, R. Saunders, and M. J. Nelson, "Semi-automated level design via auto-playtesting for handheld casual game creation," in *Proc. Comput. Intell. Games Conf.*, 2016, pp. 1–8.

[26] G. Volkmar, N. Mähmann, and R. Malaka, "Procedural content generation in competitive multiplayer platform games," in *Proc. Joint Int. Conf. Entertainment Comput. Serious Games*, 2019, pp. 228–234.

[27] A. Liapis, G. Smith, and N. Shaker, "Mixed-initiative content creation," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds., New York, NY, USA: Springer, 2016, pp. 195–214.

[28] E. Butler, A. M. Smith, Y.-E. Liu, and Z. Popovic, "A mixed-initiative tool for designing level progressions in games," in *Proc. ACM Symp. User Interface Softw. Technol.*, 2013, pp. 377–386.

[29] M. Van Kreveld, M. Löffler, and P. Mutser, "Automated puzzle difficulty estimation," in *Proc. Comput. Intell. Games Conf.*, 2015, pp. 415–422.

[30] F. Southey, G. Xiao, R. C. Holte, M. Trommelen, and J. W. Buchanan, "Semi-automated gameplay analysis by machine learning," in *Proc. Artif. Intell. Interactive Digit. Entertainment Conf.*, 2005, pp. 123–128.

[31] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin, and Z. Popovic, "Evaluating competitive game balance with restricted play," in *Proc. Artif. Intell. Interactive Digit. Entertainment Conf.*, 2012, pp. 26–31.

[32] P. García-Sánchez, A. Tonda, A. M. Mora, G. Squillero, and J. J. Merelo, "Automated playtesting in collectible card games using evolutionary algorithms: A case study in Hearthstone," *Knowl.-Based Syst.*, vol. 153, pp. 133–146, 2018.

[33] V. Volz, G. Rudolph, and B. Naujoks, "Demonstrating the feasibility of automatic game balancing," in *Proc. Genet. Evol. Comput. Conf.*, 2016, pp. 269–276.

[34] A. Zook, B. Harrison, and M. O. Riedl, "Monte-Carlo tree search for simulation-based strategy analysis," in *Proc. Found. Digit. Games conf.*, 2015, *arXiv:1908.01423*.

[35] F. de Mesentier, S. Silva, J. Lee Togelius, and A. Nealen, "AI as evaluator: Search driven playtesting of modern board games," in *Proc. AAAI Workshops*, 2017, pp. 959–966.

[36] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Evolving card sets towards balancing dominion," in *Proc. Evol. Comput. Congr.*, 2012, pp. 1–8.

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

[38] O. Keehl and A. M. Smith, "Monster Carlo 2: Integrating learning and tree search for machine playtesting," in *Proc. IEEE Conf. Games*, 2019.

[39] M. Morosan and R. Poli, "Automated game balancing in Ms PacMan and Starcraft using evolutionary algorithms," in *Proc. Eur. Conf. Appl. Evol. Comput.*, 2017, pp. 377–392.
[40] R. Leigh, J. Schonfeld, and S. J. Louis, "Using coevolution to understand and validate game balance in continuous games," in *Proc. Genet. Evol. Comput. Conf.*, 2008, pp. 1563–1570.
[41] G. N. Yannakakis and J. Hallam, "Evolving Opponents for Interesting Interactive Computer Games," in *Proc. Intl. Conf. Simulation Adaptive Behavior*, MIT Press, 2004.
[42] G. N. Yannakakis and J. Hallam, "A generic approach for generating interesting interactive Pac-Man opponents," in *Proc. IEEE Symp. Comput. Intell. Games*, Colchester, 2005, pp. 94–101.
[43] G. N. Yannakakis and J. Hallam, "Real-time adaptation of augmented-reality games for optimizing player satisfaction," in *Proc. IEEE Symp. Comput. Intell. Games*, 2008, pp. 103–110.
[44] T. W. Malone, "Heuristics for designing enjoyable user interfaces: Lessons from computer games," in *Proc. Conf. Hum. Factors Comput. Syst.*, 1982, pp. 63–68.
[45] J. Togelius, R. De Nardi, and S. M. Lucas, "Making racing fun through player modeling and track evolution," in *Proc. SAB'06 Workshop Adaptive Approaches Optimizing Player Satisfaction Comput. Physical Games*, 2006.
[46] G. N. Yannakakis, H. H. Lund, and J. Hallam, "Modeling children's entertainment in the playware playground," in *Proc. IEEE Symp. Comput. Intell. Games*, 2006, pp. 134–141.
[47] S. F. Gudmundsson *et al.*, "Human-like playtesting with deep learning," in *Proc. Comput. Intell. Games Conf.*, 2018.
[48] A. M. Smith, C. Lewis, K. Hullet, G. Smith, and A. Sullivan, "An inclusive view of player modeling," in *Proc. Found. Digit. Games Conf.*, 2011, pp. 301–303.
[49] J. Pfau, "Dungeons & Replicants," Jul. 2021. [Online]. Available: osf.io/3ktc6
[50] J. Pfau, "Deep player behavior modeling," Ph.D. dissertation, Dept. Math. Comput. Sci., Universität Bremen, Bremen, Germany, 2021. [Online]. Available: https://doi.org/10.26092/elib/727
[51] J. Pfau, J. D. Smeddinck, and R. Malaka, "Deep player behavior models: Evaluating a novel take on dynamic difficulty adjustment," in *Proc. Extended Abstr. CHI Conf. Hum. Factors Comput. Syst.*, 2019, pp. 1–6.
[52] G. N. Yannakakis, P. Spronck, D. Loiacono, and E. André, "Player modeling," in *Artificial and Computational Intelligence in Games. Dagstuhl Follow-Ups*, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds., Wadern Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013, vol. 6, pp. 45–59.
[53] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.
[54] R. Hunicke, "The case for dynamic difficulty adjustment in games," in *Proc. ACM SIGCHI Int. Conf. Adv. Comput. Entertainment Technol.*, 2005, pp. 429–433.