Diversifying Dynamic Difficulty Adjustment Agent by Integrating Player State Models into Monte-Carlo Tree Search

JaeYoung Moon[a] (super_moon@gm.gist.ac.kr), YouJin Choi[b] (chldbwls304@gist.ac.kr), TaeHwa Park[b] (taehwa-p@gm.gist.ac.kr), JunDoo Choi[a] (jundoochoi@gm.gist.ac.kr), Jin-Hyuk Hong[b] (jh7.hong@gist.ac.kr), Kyung-Joong Kim[b] (kjkim@gist.ac.kr)

[a] Dept. of Artificial Intelligence Graduate School, Gwangju Institute of Science and Technology, 123, Cheomdangwagi-ro, Buk-gu, Gwangju, Republic of Korea, 61005

[b] Dept. of School of Integrated Technology, Gwangju Institute of Science and Technology, 123, Cheomdangwagi-ro, Buk-gu, Gwangju, Republic of Korea, 61005

Corresponding Author:

Kyung-Joong Kim

Dept. of School of Integrated Technology, Gwangju Institute of Science and Technology, 123, Cheomdangwagi-ro, Buk-gu, Gwangju, Republic of Korea, 61005

Tel: 82+(62)-715-5345

Email: kjkim@gist.ac.kr

Diversifying Dynamic Difficulty Adjustment Agent by Integrating Player State Models into Monte-Carlo Tree Search

JaeYoung Moon[a,1], YouJin Choi[b,1], TaeHwa Park[b], JunDoo Choi[a], Jin-Hyuk Hong[b], Kyung-Joong Kim[b,*]

[a]*Dept. of Artificial Intelligence Graduate School, Gwangju Institute of Science and Technology, 123, Cheomdangwagi-ro, Buk-gu, Gwangju, Republic of Korea, 61005*
[b]*Dept. of School of Integrated Technology, Gwangju Institute of Science and Technology, 123, Cheomdangwagi-ro, Buk-gu, Gwangju, Republic of Korea, 61005*

Abstract

Game developers have employed dynamic difficulty adjustment (DDA) in designing game artificial intelligence (AI) to improve players' game experience by adjusting the skill of game agents. Traditional DDA agents depend on player proficiency only to balance game difficulty, and this does not always lead to improved enjoyment for the players. To improve game experience, there is a need to design game AIs that consider players' affective states. Herein, we propose AI opponents that decide their next actions according to a player's affective states, in which the Monte–Carlo tree search (MCTS) algorithm exploits the states estimated by machine learning models referencing in-game features. We targeted four affective states to build the model: challenge, competence, valence, and flow. The results of our user study demonstrate that the proposed approach enables the AI opponents to play automatically and adaptively with respect to the players' states, resulting in an enhanced game experience.

Keywords: Game artificial intelligence, dynamic difficulty adjustment, Monte-Carlo tree search, human-centered, machine learning, player modeling

*Corresponding author.
Email addresses: super_moon@gm.gist.ac.kr (JaeYoung Moon), chldbwls304@gist.ac.kr (YouJin Choi), taehwa-p@gm.gist.ac.kr (TaeHwa Park), jundoochoi@gm.gist.ac.kr (JunDoo Choi), jh7.hong@gist.ac.kr (Jin-Hyuk Hong), kjkim@gist.ac.kr (Kyung-Joong Kim)

[1]J. Moon and Y. Choi contributed equally to this work.

1. Introduction

A balanced challenge is an important factor for the satisfaction of players' gaming experience (Csikszentmihalyi, 1991; Chen, 2007; Cowley et al., 2008; Ang & Mitchell, 2017). Game developers use predefined curves that manipulate the level of difficulty as they undergo trial and error, but developing the curves is a time-consuming and challenging task due to differing players' needs (Ang & Mitchell, 2017). Therefore, many studies (Hunicke, 2005; Xue et al., 2017; Frommel et al., 2018; Zohaib & Nakanishi, 2018; Ang & Mitchell, 2019; Constant & Levieux, 2019) have employed dynamic difficulty adjustment (DDA) methods to automatically control game components to improve each unique player experience (PX). DDA methods measure how well a player plays a game through a fitness function (herein, called score function) and manipulate game components, such as game parameters (game speed), game environment (next block of Tetris), or the game AI itself (beginner-level or high-level strategies of agents in fighting games), to adjust the difficulty. As AI technology advances, there has been a growing interest in direct game AI manipulation for DDA methods to achieve more effective game difficulty adjustment (Demediuk et al., 2017; Zohaib & Nakanishi, 2018; Ishihara et al., 2018; Pfau et al., 2020; Moon & Seo, 2020). Most of these studies concentrate on adapting game components for game difficulty by focusing on the player's proficiency.

Many heuristic approaches have been employed to determine a player's proficiency, such as measuring win rate (Sarkar & Cooper, 2019; Duque et al., 2020), game score (Hagelback & Johansson, 2009; Silva et al., 2017), and health points (HP) (Demediuk et al., 2017; Ishihara et al., 2018). For example, games with opponent agents, such as fighting games, adjust the difficulty by changing the skills of the game agents using tree search (Demediuk et al., 2017; Ishihara et al., 2018) or deep learning (Pfau et al., 2020), according to heuristic measures like HP-difference. Independent of the model used, if the model is based on a simple heuristic score function, the gameplay is relatively static and has predictable

difficulty adaptations (Hunicke, 2005; Ang & Mitchell, 2017; Constant & Levieux, 2019; Stephenson & Renz, 2019). Players have certain difficulty expectations from games. For instance, some players may prefer challenging games, whereas some enjoy more relaxed, easily winnable games. Simple heuristic measurements are often too limited to satisfy the wide breadth of player expectations.

Some researchers have developed DDA models to estimate players' affective states using sensors (Rani et al., 2005; Tognetti et al., 2010; Kivikangas et al., 2011; Chanel et al., 2011; Kneller et al., 2012; Afergan et al., 2014; Parnandi & Gutierrez-Osuna, 2015; Fernandez B. et al., 2017; Stein et al., 2018). Instead of measuring player proficiency, they focus on how their DDA model improves the actual PX. However, they employed simple adjustment mechanisms, which manually controlled game parameters, such as game speed. In this case, the affective model to estimate player state is outside the adjustment mechanism and does not change according to the individual player. Thus, the game difficulty does not accurately match the player's estimated state, but rather operates based on a pre-designed function designed by the developer. Therefore, it is necessary to apply the player states directly to the game AI mechanism, and automatically generate strategies to satisfy players' complex preference instead of manual design.

To easily and precisely control the game difficulty and satisfy players' diverse preferences we designed a DDA-based game AI mechanism that directly considers various players' affective states and adjusts the game difficulty according to the recognized state. The player state model based on machine learning (ML) methods predicted the affective state of a game player, and the Monte–Carlo tree search (MCTS) algorithm (Chaslot et al., 2008) exploited the ML model as its scoring function. We trained four ML models for four states, including challenge (CH), competence (CO), valence (VA), and flow (FL). Instead of relying on external sensors, our player state model estimates player's states only using in-game features (or logs). To evaluate the performance of the proposed approach and the effect on PX, we conducted a user study with 20 human players, where our game agent played diverse action strategies according to each player's state

and improved their PX.

The contributions of this study are as follows:

- This is the first attempt to incorporate the player's affective states directly into DDA agents and automatically generate game AI appropriate for a specific player state.
- Game AI with diverse skill strategies was generated to meet a wide breadth of players' preferences and improve the PX.

2. Background and Related Work

Dynamic Difficulty Adjustment (DDA) consists of two steps (Zohaib & Nakanishi, 2018). First, measuring the players' skill levels through a score function. Then, adjusting the game difficulty in various ways according to the measured players' skill levels. Section 2.1 demonstrates the DDA methods which mainly concentrated on the difficulty adjustment methods (second step), and Section 2.2 introduces the DDA methods which mainly concentrated on the player modeling for measuring players' level (first step) (summarized in Table 1).

2.1. Dynamic Difficulty Adjustment

Many DDA studies have mainly focused on game difficulty adjustment methods, including game parameter tuning (Denisova & Cairns, 2015), game environment manipulation (Lora et al., 2016; Duque et al., 2020), and game agents (Olesen et al., 2008; Silva et al., 2017; Demediuk et al., 2017; Ishihara et al., 2018; Pfau et al., 2020), to control game difficulties. As related to in-game parameter tuning, Denisova & Cairns (2015) reported that PX was increased through dynamic adjustment of game difficulty using a simple timer depending on the player's skill. For in-game environment manipulation, Hunicke & Chapman (2004) adjusted the game level by systemically regulating item drop rate (rule-based) according to users' inventory expenditure and damage patterns. They assumed that the inventory status and damage pattern could represent players' skills.

More recently, game AI agents for a complex difficulty adaptation have been the most actively studied. Demediuk et al. (2017) and Ishihara et al. (2018) reported an MCTS based DDA agent method for a fighting game. They manipulated the game difficulty by implementing a DDA agent that uses the HP difference between a player and an agent for the score function using an MCTS algorithm. Pfau et al. (2020) demonstrated a DDA agent using a deep learning technique for the commercial game, AION. They proposed an AI non-player character, which imitates player behavior every time an instance dungeon is played and hypothesized that it could play similarly to players. Like these examples, several DDA studies on adjusting functions with AI have been conducted.

Most DDA techniques measure player skills using heuristic score functions specified by game developers. These methods assume that adjusting game difficulty by heuristically measuring players (through HP, game score, damage, *etc.*) can increase the PX. However, PX and game difficulty adjusted by heuristic measures are often inconsistent (Hunicke, 2005; Ang & Mitchell, 2017; Constant & Levieux, 2019; Stephenson & Renz, 2019).

Table 1 A comparison table of related works on DDA.

Author(s)	Game Genre	Measuring the player's level (first step)	Adjusting difficulty (second step)
Denisova & Cairns (2015)	Shooting	HM (Game score)	PT (Timer)
Hunicke & Chapman (2004)	FPS	HM (Item consumption)	EM (Item drop pattern)
Duque et al. (2020)	Zelda-like	HM (Win rate)	EM (Map generation)
Demediuk et al. (2017)	Fighting	HM (HP difference)	**GAI (MCTS)**
Ishihara et al. (2018)	Fighting	HM (HP difference)	**GAI (MCTS)**
Pfau et al. (2020)	MMORPG	HM (Player behavior)	**GAI (Deep learning)**
Fernandez B. et al. (2017)	Rhythm game	**AM (Emotion)**	EM (Rhythm generation)
Afergan et al. (2014)	Flight simulation	**AM (Cognitive workload)**	EM (UAV route control)
Parnandi & Gutierrez-Osuna (2015)	Racing	**AM (Emotion)**	PT (car speed, *etc.*)
Chanel et al. (2011)	Tetris	**AM (Emotion)**	PT (block speed)
Frommel et al. (2018)	Space jump	**AM (Emotion)**	PT (number of obstacles, *etc.*)
Stein et al. (2018)	Shooting	**AM (Emotion)**	PT (speed, *etc.*)
Tognetti et al. (2010)	Racing	**AM (Preference)**	PT (car speed)
Liu et al. (2009)	Pong	**AM (Anxiety)**	PT (paddle speed, *etc.*)
Ours	Fighting	**AM (In-game features)**	**GAI (MCTS)**

Notes. Measuring the player's level (first step); Heuristic method (HM) and Affective Modeling (AM). Adjusting difficulty (second step); Parameter Tuning (PT), Environment Manipulation (EM), and Game Artificial Intelligence (GAI).

2.2. Player State in DDA

Game difficulty can evoke various cognitive and emotional states that can impact PX (Kneller et al., 2012; Kivikangas et al., 2011). Some studies have been conducted to directly deal with PX and apply DDA by estimating the players' mental states, such as cognitive load (Afergan et al., 2014), emotion (Chanel et al., 2011; Parnandi & Gutierrez-Osuna, 2015; Fernandez B. et al., 2017; Frommel et al., 2018; Stein et al., 2018), preference (Tognetti et al., 2010), and anxiety (Rani et al., 2005; Liu et al., 2009). Afergan et al. (2014) applied a workload detection model using EEG sensors to determine probability value of difficulty adjustments for probability-based flight paths in flight training games. Parnandi & Gutierrez-Osuna (2015) proposed an EDA-sensor-based arousal classification model to adjust difficulty through adjusting the speed of racing games. They found that car speed is the main determinant of player arousal in racing games. Chanel et al. (2011) also confirmed that players' emotions, estimated from physiological signals, can be used as an indicator to adjust the speed of falling blocks in Tetris for DDA.

Most of these studies have focused on developing a player mental-state prediction model to estimate the abilities of players through external factors, such as sensors and surveys. To adjust game difficulty, relatively straightforward methods, such as tuning game parameters, have been used. As mental-state prediction models are used outside of the adjustment mechanism, game developers are limited to designing preset strategies to improve PX. Conversely, our approach directly employs player states to the game AI for DDA to automatically and progressively adjust AI opponent actions.

2.3. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is the most frequently used method in game AI field. It searches the AI's semi-optimal next action by simulating tremendous of possible situations from the current game state. It consists of four steps: selection, expansion, simulation, and backpropagation. In every calculation, the current state is set as a root node. Then, the four steps for the

tree search operate within a set time limit. After completing the tree search, the agent chooses the most visited child node or most valuable child node for the next action.

- **Selection (Figure 1B.1):** In every node from the root to the expanded leaf, the most valuable child node is selected by several policies. Herein, we employed the upper confidence bound (UCB1; Eq. 1) method (Auer, 2003), the most popular policy in MCTS. Where C is a constant for balancing the exploitation and exploration of the visiting of children nodes, n_i is the number of trials of i-th child node, N is the total number of trials, and v_i (Eq. 2) is a value estimation term of the i-th child node. UCB1 can be computed after every child node is visited at least once. Until every child node is visited, the child node is uniformly selected. Otherwise, a child who has the largest UCB1 value is selected as the next node to visit.

$$UCB1_i = v_i + C\sqrt{\frac{2lnN}{n_i}} \tag{1}$$

$$v_i = \frac{\sum^{n_i}(scores)}{n_i} \tag{2}$$

- **Expansion (Figure 1B.2):** It expands the tree toward the terminal state. If a leaf node is visited more than the predefined threshold V_i which indicates the maximum number of visitation of i-th node, and the depth of the node does not reach the predefined threshold D_i which indicates the maximum expansion-depth of i-th node, the children of the current node are expanded. If the depth of a leaf node from the root node is equal to D_i, the node will not expand its children anymore.

- **Simulation (Figure 1B.3):** If a leaf node is not visited enough, the simulation step operates. In this step, MCTS performs a sequence of random actions for the agent and the opponent until the predefined simulation timeout is reached. At the end of the simulation, the score function

calculates the value of the nodes explored. The score function is defined depending on the application. Herein, the randomly simulated actions (nodes) are called as trajectory.

- **Backpropagation (Figure 1B.4):** Finally, when the simulation is over, the computed score is updated as values (Eq. 2) for every traveled node in the trajectory. While the value is updated, the number of visitation N for each visited node increases simultaneously.

3. Architecture of the DDA Model for an AI Opponent

Herein, we show the development of our model. Figure 1 shows the overall architecture of the model, which incorporates an MCTS algorithm and a player state prediction model. The main difference between our model and the heuristic DDA method is that we directly used players' state as a score function for MCTS instead of using heuristic functions, such as game score, win rate, and HP differences. The player state model uses a combination of the real-played and simulated game logs by MCTS to predict how the player state will change in the future. The player state model is trained using ML methods to predict the players' state in real-time using game logs. MCTS performs the action selection for the DDA through comprehensive random simulations until the most visited action is selected as the next action.

3.1. Player State Model

A player state model is employed as a score function to evaluate each value (Eq. 2) The player state model estimates a player's certain state when the DDA agent follows the simulated game scenario. To diversify the strategy of DDA agents, several player state models evaluate various player states. The DDA agent's strategy will vary depending on which player state model is used to improve the target player state.

The player state model is trained by machine learning manners using [game log - player state] data pair. It exclusively uses in-game components (game log)

as input to the model and predicts the target player state based on the input game logs. The game log consists of a few consecutive seconds of game frames. We recruited human players to play the game to collect the [game log - player state] dataset, with the aim of using DDA. The participants played the game against various AI levels to observe the players' responses to different game conditions. After each game, the participants completed a questionnaire on their game experience. By analyzing the survey result, the player state corresponding to the game log is annotated for the [game log - player state] data pairs.

Herein, the player state model is trained as a binary classifier that predicts how the player is classified for the target state. Rather than applying solely the classification results to evaluate the nodes in MCTS, we used probability factors (i.e., confidence about the result of the model) for the classification results to score. For example, if a player state model predicts the probability for a player's target state engagement of 0.75, the classification result for the state will be 1, but we used 0.75 for the score. On the contrary, if the prediction for the probability of the engagement of the state is 0.25, the classification result will be 0, but we used -0.75 for the score.

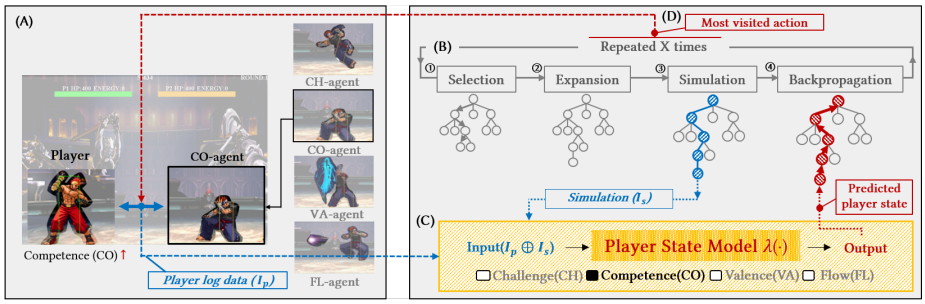
3.2. Monte Carlo Tree Search Integrated with Player State Model

The setting of MCTS for our model differs significantly from the original MCTS in two ways. First, we used the player state model to score the simulated trajectory, unlike standard approaches that use the direct result of the MCTS, such as game score, win rate, and HP-difference for the score. Second, to give a time-series input for the player state model, the simulated frames are accumulated in memory while the simulation propagates, unlike typical MCTS methods that simply use the leaf node data. Details of our DDA process are as follows:

At each frame in the game, the DDA agent accumulates the game log (Figure 1A, I_p); the game log has various game features, which represent the game condition described in Table 2. The agent performs MCTS iterations discussed in Section 2.3 to search for the next action (Figure 1B). For each

iteration, game features of the simulated trajectory (I_s in Figure 1) are logged and concatenated behind the cumulated real-played log data ($I_p \oplus I_s$ in Figure 1C, where $\oplus$ is concatenation). The player state model predicts the players' state from a combination of the real and simulated game log. The statistics of the MCTS nodes in the simulated trajectory are updated by the probability of the predicted state (Figure 1C). After MCTS times out, as defined by the system developer, the most visited child node is selected as the next action (Figure 1D).

Figure 1 Overall architecture of the Monte–Carlo tree search (MCTS) algorithm exploiting player states. (A) Gameplay scenes of Player vs. four AI styles of the dynamic difficulty adjustment (DDA); (B) MCTS process to select the next DDA agent action; (C) Player-state prediction models built using machine learning algorithms, which take real game history (I_p) and simulated game logs (I_s) as an input; (D) Next selected action, which is the most visited in the MCTS process.



4. Implementation Details for Experiment

4.1. Environmental Settings

We employed a fighting game for our experiment, which is a representative genre in which players play the game against AI opponent. There are four player states we aimed to encourage; Challenge (CH), Competence (CO), Valence (VA), and Flow (FL) which were derived from the Game Experience Questionnaire (GEQ) survey (IJsselsteijn et al., 2013). MCTS time limit for the entire searching calculation was set as 165 ms to meet the real-time performance of the agent. In other words, the four steps of the MCTS (2.3) iterated as many times as possible within 165 ms. The constant values of C, V_i, and D_i for the MCTS were set as

3, 4, and 3 respectively. The simulation step of the MCTS (2.3.Simulation) was operated for 0.5 sec or 30 frames (game frame rate = 60 fps). The player state model takes 5 sec of game log data as input with 4.5 sec of real played logs and 0.5 sec of simulated logs from MCTS.

There are 56 actions in the Fighting ICE game platform (Lu et al., 2013) grouped into six. NO_OP means the agent did nothing, which implies it maintains the current state. It contains seven actions, including neutral, stand, air, change down, down, rise, and landing. OUT_OF_CONTROL contains six actions that are unable to control the agent, like stand guard recovery. MOVE consists of seven moving actions, including forward walk, back step, and jump. NORMAL_ATTACK is a group of 20 simple attack commands that combine an attack key (punch, kick) with at most one arrow key (up, down, left, or right). SPECIAL_ATTACK contains 13 complex commands that combine attack keys (punch, kick) with arrow keys (up, down, left, and right) in certain sequences and expends energy with each use. Finally, the GUARD group contains three guard actions, including stand, crouch, and air guards.

4.2. Input and Output for The Player State Model

The abbreviated features are listed in Table 2. The total number of features per frame is 63. The player state model takes 5 sec of data as input to predict a player's state. At a frame rate of 60 fps, the model receives too many features ($300 \times 63 = 18{,}900$) as input if using 5 s of frames. This inhibits the search performance of MCTS, which fools the agent. By sampling the frames at intervals of 15 frames, the average frames that a single action is performed, we reduced the input size from 18,900 to 1,260 (20×63). The output of the player state model is the probability of the player to be classified into the state.

4.3. Data Collection

To build a classifier that predicts the state from the game log data, we employed [game log – player state] data pairs. To collect the training dataset, we chose three AI level agents to compete against a human player from the

Table 2 Abbreviated input features of the player state prediction model. Each feature is logged by players: human player and AI opponent, except common features. In total, there are 63 features per each frame.

Common Features (count)	Character Features of player & agent (count)	Attack & Projectiles Features of player & agent (count)
x, y distance of characters(2)	x, y positions(4)	attack type(4)
HP difference(1)	facing direction(2)	hit damage(4)
	state id(2), action id(2)	guard damage(4)
	speed to x, y directions(4)	knock-back distance to x, y directions(8)
	HP(2), energy(2)	x, y distance from the target(8)
	approaching to the opponent(2)	speed to x, y directions(8)
	remaining frames to be neutral state(2)	the number of projectiles(2)

16 AIs available on the official GitHub page for Fighting ICE. The submitted agents were ranked by win rate after the league. By ranking, the agents were divided into three groups: upper (five agents), middle (six agents), and lower (five agents). We conducted a pilot test for each group from five game experts who have over three years of experience in game research. The experts played games against agents in each group and then scored a 5-point Likert-type scale ranging from 1 (Not at all) to 5 (Very much) to the question, *"Does the opponent player look like a person?."* We selected three agents with the highest score in each group; Reiwa Thunder (in the upper group, M = 4.2, SD = 0.84), HaibuAI (in the middle group, M = 4.6, SD = 0.55), and TOVOR (in the lower, M = 4.2, SD = 0.45). Additionally, we added the lowest level "keyboard" agent, which made no movements, to observe the players' state when the opponent was not moving. Then, using these four agents, we conducted data collection from 43 participants. Each participant played four games per AI agent. At the end of each game, a GEQ survey was conducted. The survey list was reorganized by selecting the questions appropriate for the fighting game. We assembled 688 game logs (43 participants $\times$ 4 AI agents $\times$ 4 games) and GEQ survey data for each of the 688 games played.

4.4. Model Building and Validation

By analyzing the results of the GEQ survey, we labeled whether each game engaged a state (CH, CO, VA, or FL) for each player. Using these labels and logged gameplay data, we trained the player-state classification models using

various ML methods. Though classification was performed with 5 sec of data, there was only one label per game. We hypothesized that a portion of the game could represent the entire game due to the nature of the fighting game, which has a very short length and does not change dramatically throughout. Therefore, we fragmented the game at intervals of 5 sec with strides of 2 sec and then matched the label to the fragments. For example, if a game was played for 1 min, there would be 27 data fragments from the game. As mentioned in Section 4.2, the total input size was 1,260 inputs per game. The result of the 10-fold cross validation of each trained model is shown in Table 3. We employed a light gradient boosting machine (Light GBM) model (Ke et al., 2017) to predict for CH, CO, and VA, and a random forest model (Breiman, 2001) to predict for FL in the MCTS algorithm.

Table 3 10-fold cross validation results of the player state classification.

ML algorithm	Challenge (CH)	Competence (CO)	Valence (VA)	Flow (FL)
Logistic Regression	69.2%	66.7%	66.3%	71.8%
SVM	69.5%	67.5%	67.9%	72.4%
LightGBM	**71.5%**	**69.4%**	**68.4**%	72.2%
Random Forest	61.1%	69.1%	68.3%	**73.1**%
ZeroR	53.3%	53.7%	53.3%	53.5%

The simplest classification approach is ZeroR, which simply predicts majority category. It is mostly useful for establishing baseline performance as a benchmark for other classification methods. We employed DummyClassifier in the scikit-learn library (Pedregosa et al., 2011) for computing ZeroR

5. User Study

5.1. Materials

A counter-balanced single factor within-subjects design was conducted to validate the performance and PX of our DDA model compared to the current heuristic DDA models. For our model, four game agents were designed each for CH (CH-agent), CO (CO-agent), VA (VA-agent), and FL (FL-agent), which is the principal dimension of PX. Each agent played the game aimed to elevate the players' target state. We employed prior work by Demediuk et al. (2017) as a baseline (HP-agent), as often exploited in many DDA studies (Ishihara et al.,

2018; Kusano et al., 2019; Liu et al., 2020). This work aims to try to reduce the HP difference between the player and the agent within a threshold. Since the HP-agent consisted of an MCTS algorithm (like ours) but exploited a simple heuristic score function (HP-difference) as opposed to ours, we assumed that it could be a representative example of the difference between the traditional DDA method and our novel approach. We set the threshold to max HP $\times$ 0.1, mirroring the prior work; our game's max HP is 400, so the threshold was set to 40 HP. The HP-agent performed somewhat static actions to maintain this threshold. When the absolute difference of HP between the player and the agent was within the threshold (40), the agent performed actions, such as jumping in place, guarding and attacking regardless of the player's position. However, if the HP difference crossed the threshold, the agent rushed to attack the player.

5.2. Measurements

To score each participant's PX, we employed a GEQ designed to provide a comprehensive evaluation of the gameplay experience. Many studies on games have used this questionnaire, including psychophysical studies of PX (Kuikkaniemi et al., 2010; Nacke et al., 2010). By examining many factors of PX, we filtered the survey into subscale components to match the fighting game used for this study. The questionnaire is designed with five subscales: Challenge (4 items:18–21), Competence (4 items: 1–4), Positive effect (4 items: 9–12), and Negative effect (5 items: 13–17), Flow (4 items: 5–8), with each using a seven-point scale. Challenge indicates how challenged a player feels. Competence measures a player's ability to reach game goals. Positive and negative effects describe positive and negative emotions during gameplay. The sum of Positive effect score and a reverse score of Negative effect is considered as Valence state factor in this study. Flow is related to the sensation of playing within the confines of the game.

Herein, the sum of the subscale factors was used as the total PX score, which implies a player's preference for a game, and each subscale component of GEQ was used as a basis for each state score (CH, CO, VA, and FL) that indicates how

a player felt the state when they played a game that our DDA agent targeted to enhance.

5.3. Participants

20 college and graduate students (12 males, 8 females) aged between 23 and 32 (M = 26.15, SD = 3.09) were recruited for this study to play the fighting game. Half of the participants had an average of more than 10 hours of gameplay, whereas others usually spent less than 1 hour per week. All participants were experienced in playing fighting games.

5.4. Procedure

Each participant session lasted about 60 min and was divided into two parts: preparation and experimental phases. In the preparation phase, participants were asked to sign a consent form and complete a demographic questionnaire, after which they practiced the fighting game. Participants were familiarized with the basic controls (e.g., move and attack) and game mechanics (attacking the AI agent) during the tutorial session. The experimental phase consisted of two sessions, in which the participant played five games and filled in the GEQ questionnaire for each game. Games were ordered by counter-balancing using random Latin squares. A cool-down period (10 min) was given after one session to reduce the participant's fatigue. After completing the experimental phase, participants responded to interviews about the PX.

6. Analysis and Result

We collected 200 samples of the questionnaire and log data (20 participants $\times$ 5 game conditions $\times$ 2 sessions). Scale reliability tests were performed for the five subcategories of GEQ. Cronbach's alpha (Cronbach, 1951) showed strong reliability for all subcategories (Challenge: $\alpha = 0.88$; Competence: $\alpha = 0.93$; Positive effect: $\alpha = 0.96$; Negative effect: $\alpha = 0.86$; Flow: $\alpha = 0.94$), indicating consistent answers. The analysis was conducted with two main hypotheses to

analyze whether the agents create a satisfying experience, by selecting different strategies based on player state:

1) Does a player state enable DDA agents to generate/play diverse actions?

2) Does a state-specific DDA agent satisfy various PX?

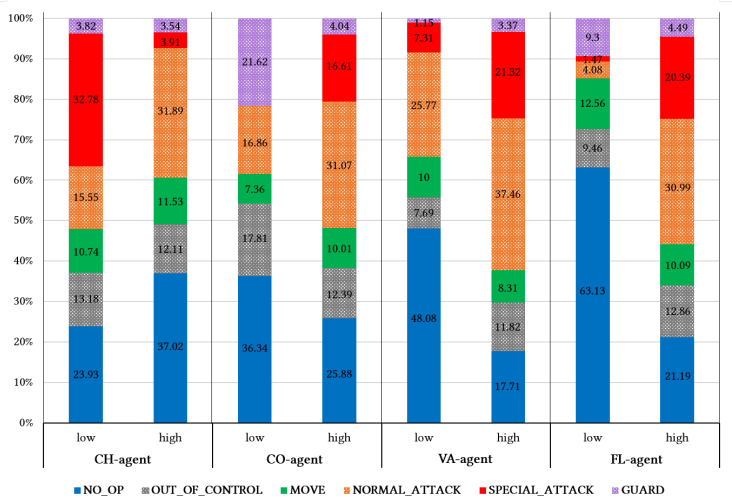
6.1. Does a Player State Enable DDA Agents to Generate/Play Diverse Actions?

We analyzed four agents that aimed to improve the corresponding player states (CH, CO, VA, and FL). Each state was estimated by ML model with 1,260 features (20 frames of real and simulated game logs × 63 features per frame) as described in Section 4.2. In this section, we don't cover the patterns of HP-agent, already discussed in Section 5.1. Figure 2 shows the distribution of the actions of each agent. Each column represents the distribution of the action groups determined by the agent when it predicts the player's state as low (<50% probability) or high (≥50% probability). Figure 3 shows a player's state graph estimated by the ML model (Figure 3A) with several actions of the agents corresponding to the player state (Figure 3B) during the game. In general, our proposed DDA approach generated diverse strategies according to players' affective states. The details are described for each player state as follows.

6.1.1. CH-agent: DDA agent that challenges players

The **CH** agent showed highly aggressive movements (*NORMAL_ATTACK* (15.55%), *SPECIAL_ATTACK* (32.78%)) when it estimated a low CH state, and vice versa for a high CH state. This was in contrast to the behavior of the other three agents. In particular, *SPECIAL_ATTACK* (32.78%) was the most frequently used action when the CH state was estimated as low. The CH agent acted aggressively to increase the CH state of the player. However, when the CH state is estimated as high, it maintains tension by reducing total attacks but increasing the *NORMAL_ATTACK* (31.89%) ratio and *NO_OP* (37.02%). However, the CH agent did not perform as much as expected (we will discuss about this in Section 6.2).

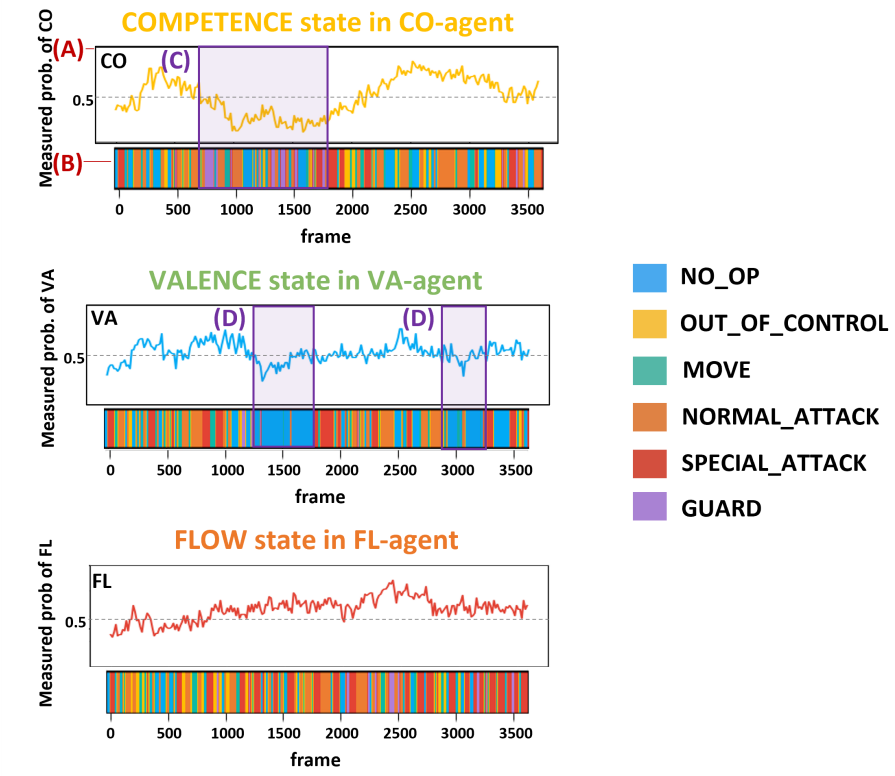
Figure 2 Action distribution of each agent. Low represents an action distribution when the agent estimates the player state as low (<50%), and high indicates the opposite case.



6.1.2. CO-agent: DDA agent that lets players show their competence

The **CO** agent played very defensively in the low state—*GUARD* (21.62%), *NORMAL_ATTACK* (16.86%), and *SPECIAL_ATTACK* (0%). We infer that when a player's CO state is estimated as low, it attempts to improve the player's competence by taking a defensive stance, such as guard and moving actions. This is shown in Figure 3C, with the agent lowering the *ATTACK* action ratio and increasing *NO_OP* and the *GUARD* ratio when the agent detects that the player's CO state is falling. Also, the agent exhibited the least frequent attacks in the high state among the three agents were (CO: 47.68%; VA: 58.78%; and FL: 51.38%; sum of *NORMAL_ATTACK* to *SPECIAL_ATTACK*) ratios (Figure 2). Overall, the CO agent lowers its offensive tactics and gives more chances to players to attack when it thinks the player in low competency. **P6**, was satisfied with the CO agent (1^{st} rank) and expressed, *"I can consider my actions more deeply in this game because it gave me more time to rest compared to other games."*

Figure 3 Samples of the performed action sequence corresponding to the estimated player state during the game. (A) Estimated state graph during a game through ML model. (B) Agent's actions corresponding to the player state estimated. (C) Agent's actions corresponding to the player's low CO state. (D) Agent's actions corresponding to the player's low VA state.



6.1.3. VA-agent: DDA agent that keeps pushing you in proper pacing

The **VA agent** showed aggressive patterns compared to the other three agents in both low and high VA state—*NORMAL_ATTACK* (25.77%) and *SPECIAL_ATTACK* (7.31%) in the low state; *NORMAL_ATTACK* (37.46%) and *SPECIAL_ATTACK* (21.32%) in the high state. The VA agent increased the proportion of passive actions while maintaining offensive actions when players' VA state is low —*NO_OP* (48.08%), *OUT_OF_CONTROL* (7.69%), *SPECIAL_ATTACK* (25.77%), and *NORMAL_ATTACK* (7.31%). When the state of valence decreased, the agent mostly performed static actions (*NO_OP*;

Figure 3D, shown in blue). The VA agent also performed intermittent attacks, as shown by the green line graph in Figure 3. The VA-agent adjusts its pace to the player to maintain a high player's VA state. **P10**, who most enjoyed playing with the VA agent, said that *"the agent suddenly rushed me whenever I tried to observe the agent's reaction. It seemed like the agent was reading my mind."*

6.1.4. FL-agent: DDA agent that knows when to be strong or gentle

The **FL agent** showed very static and defensive movements in the low FL state. Above 70% of the time in the low state it did not move—*NO_OP* (63.13%) and *OUT_OF_CONTROL* (9.46%) and most time was taken up by *GUARD* (9.3%) and *MOVE* (12.56%) actions. On the other hand, it was very aggressive in the high FL state. Though both VA and FL agents performed a high proportion of *ATTACKs*, the FL agent consistently pushed the player (Figure 3, orange line graph), unlike the VA agent that attacked intermittently. Overall, the FL agent encouraged player attacks when the player was in the low FL state by being passive and being aggressive in the high FL state to maintain the state. **P1** expressed, *"There were a lot of crazy attacks, but I stayed positive because the opponent stopped its attacks when I was frustrated."*

6.2. Does a State-specific DDA Agent Satisfy Various PX?

6.2.1. Game modeling by players' affective states

Validating the model through Friedman tests (Friedman, 1937), we found significant differences for the state scores of our CO, VA, and FL agent compared to the baseline HP agent. The mean values for the player state scores for the subscale component of GEQ are shown in Figure 4. The figure shows the average of each target state: 1) baseline agent (*HP*), 2) agents focusing on each state (*State*), and 3) the rest of the agents (the average of those excluding HP and targeted state agent, *All - (HP + State)*). The CH agent did not exhibit a significant difference in challenge scores from the HP agent, but for all other agents did, including CO (*Friedman's* χ^2 *(2, 17) = 10.5,* $p < 0.01$), VA (*Friedman's* χ^2 *(2, 17) = 13.3,* $p < 0.01$), and FL (*Friedman's* χ^2 *(2, 17)*

$= 7.3,\ p < 0.05$). The mean competence state of the CO agent is 4.97 ($SD = 0.8$), and that of the HP agent is 4.4 ($SD = 1.7$), indicating that the CO agent improved the competence state over the baseline. Post-hoc pairwise comparison using the Bonferroni correction (Bonferroni, 1936) showed that the VA and FL state exhibited a significant difference compared to the average of the rest of the agents. The mean of the VA agent is 4.7 ($SD = 0.7$), which is significantly greater than the mean score of the other two conditions: *HP* ($M = 4.0,\ SD = 1.6$) and *All-(HP + VA)* ($M = 4.5,\ SD = 0.8$). The mean of the FL agent ($M = 5.26,\ SD = 1.01$) was also higher than the FL state scores of *HP* ($M = 4.5,\ SD = 2.0$) and *All-(HP + FL)* ($M = 4.8,\ SD = 1.0$). These results indicate that our agent, which uses the player's state for action selection criteria, can adjust the game to incite the targeted player state, except for the CH agent. The result of the CH agent was lower than that of its comparison to the HP agent, which may be affected by the characteristics of the game platform. It was difficult to create a more challenging overall PX (win rate; $M = 0.67,\ SD = 0.06$) because there is a limit to the factors that could be modified for this specific fighting game.

Figure 4 Targeted player-state results (GEQ) for the HP, CH, CO, VA, and FL agents. Three targeted state scores: (1) baseline agent (HP), (2) agent focused on targeting states (CH, CO, VA, and FL), and (3) agents except for two games (All games − (HP + target state agents)). Significant differences are indicated by (*$p < 0.05$, ** $p < 0.01$, and ***$p < 0.001$)

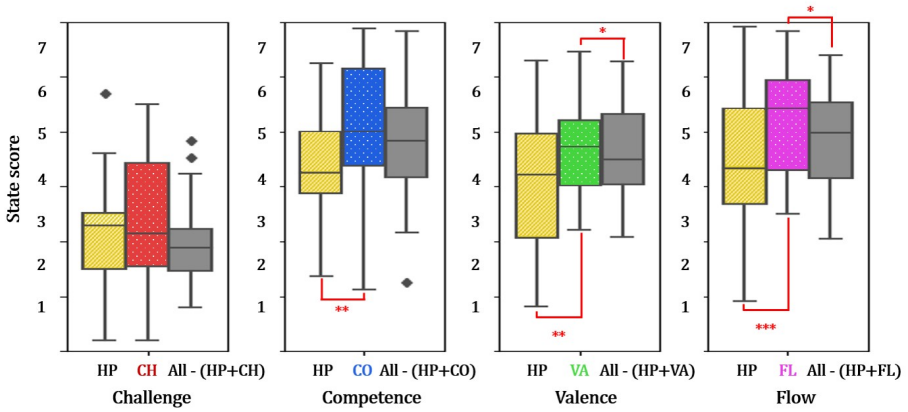


Table 4 GEQ mean score of all players. Significant improvement for satisfaction compared to HP game (*$p < 0.05$).

	Agent	Mean	SD	z	P-value
Baseline	HP	4.2	0.98		
Our Approach	CH	3.9	1.04	1.33	0.18
	CO	4.64	0.91	2.943	0.032(*)
	VA	4.62	0.93	2.432	0.014(*)
	FL	4.66	0.98	2.02	0.045(*)

6.2.2. PX by our DDA agent

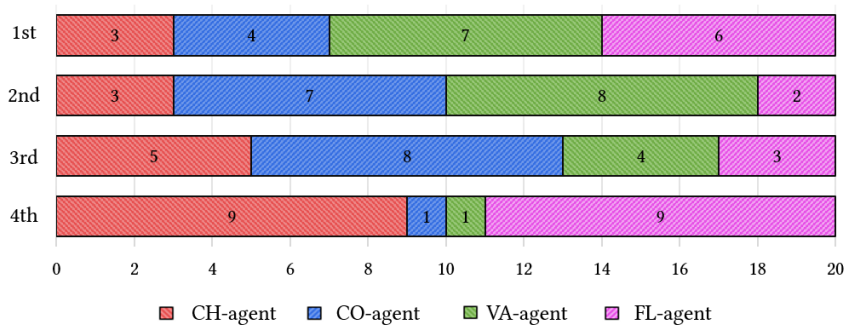
Wilcoxon signed-rank tests (Wilcoxon, 1945) were conducted to evaluate the effect of the PX according to the targeted states between the baseline agent and each state-specific agent. The score for the PX is the average of the sum of GEQ surveys, as mentioned in Section 5.2. Table 4 shows that the CO agent ($z = 1.33$, $p < 0.05$) had a significantly higher PX than the HP agent. VA ($z = 2.4$, $p < 0.05$) and the FL ($z = 2.02$, $p < 0.05$) agent also significantly improved PX compared to the HP agent. This result indicates that applying game adjustments according to a player's internal state does improve PX, achieved by our novel agents. The CH agent failed to significantly improve PX, as it also failed to provoke its intended state (Figure 4).

7. Discussion

7.1. Player Preference for AI Opponents

We found that individual player preferences for state-specific DDA agents were varied. Figure 5 exhibits the diversity of the preferences. The overall preferences were 15% (three participants) for CH, 20% (four participants) for CO, 35% (seven participants) for VA, and 30% (six participants) for FL, indicating diverse preferences for the four agents. Players' second and third preference rankings were evenly distributed. The preference ranking for each agent was influenced by individual preferences, as stated in Section 6.1. For example, some had positive opinions for the FL agent, stating that it incited immersion by constantly attempting to move and attack, but some players felt negatively about the FL agent.

Figure 5 Preferred game models ranked by players. The player's agent preference ranking was calculated as the average of two sessions, and the average rank difference between the sessions was 1.43. The Y-axis represents the first, second, third, fourth rank of the preferred game, and the X-axis represents the count of participants.



We further compared players with similar state scores that expressed differing preferences for the state-specific DDA agents. P13 and P18 scored similar competence state scores for the CO agent, but expressed differing rankings for gameplay satisfaction. P13 favored the CO agent, with a competence state score of 6.8, whereas P18 expressed lowest satisfaction with the CO agent, despite having a similar state score of 6.4. P18 preferred the CO agent. P13 responded positively to the actions of the CO agent, stating the following: *"I was satisfied because I felt accomplished and had successful attacks."* On the other hand, P18 was not satisfied, stating the following: *"I prefer difficult and aggressive opponent agents. It felt difficult initially, but suddenly, the opponent agent was too easy to attack. It felt so easy that I wasn't satisfied."* P5 and P12 showed similar FL scores for the FL agent (*flow state: P5-5.4, P12-4.6*), but their satisfaction rankings were different. P5 preferred the FL agent the most, whereas P12 ranked it fourth. P5 gave the following answer: *"There were many attacks, but I stayed positive because the opponents moved and attacked properly."* P12, on the other hand, replied, *"I think I was distracted by too many attacks, and it was difficult. I liked agents that gave me the right timing rather than frequent attacks."* As

in the example of these players, each agent showed an average of eight pairs of players with similar state scores, but different preferences. Although players had similar player state scores for a target state, the corresponding rankings were different. Creating a diversified gaming experience is essential because players' preferences are widely varied. These AI opponents herein can induce various PXs by applying state metrics and maximize satisfaction by strengthening the preferred game states.

7.2. Traditional Game Agent Design vs Our Agent Design Approach

The action strategy of DDA agents should be more personalized based on player state and preference. Previous research has pre-defined characteristics, such as aggressive or defensive based on heuristic knowledge (Ishii et al., 2018; Oh et al., 2021). These processes rely on the game developer knowledge and a repetitive design loop. They repeat a design-development-evaluate loop until a desired experience for the end user is achieved (Stahlke et al., 2020). On the other hand, our approach allows developers to dynamically create diverse styles of DDA agents by directly exploiting the player state into the AI model. This can greatly reduce playtesting. For the traditional method, if a developer creates agents with N-characteristics, N-heuristic functions and test processes is required, but since our method directly observes the player state, this lengthy process is not required. Furthermore, our method can be applied to a wide variety of games given that gameplay data and player state can be collected. Overall, our novel state-specific DDA agent is a simple, yet adaptable solution for the improvement of PX for a wide variety of games and applications.

7.3. Limitations and Future Work

There are some limitations to the study. First, the proposed approach employs a simply applicable player-state prediction model based on game logs. This is a specific data collection process to create a state model, and the performance of the player state model is not so high. Technology should be advanced so that performance can be improved and applied generally through camera-based

facial expression and behavior analysis. Second, the game used in this is simple. Therefore, player enjoyment was constrained by the simplicity of the game, affecting some outcomes, such as for our CH agent. Only one case of the Fighting game has been identified, but the proposed method may be more effective if applied to more complex commercial games. For future work, the proposed model will be applied to different game genres (e.g., role playing game, first person shooter game), various characters (e.g., other champions in the same game), and AI models (e.g., Reinforcement, Deep learning) and validated for several player groups (e.g., kids, older adults) (Mourato et al., 2014). We will also expand the subcategories of GEQ to emotion and workload states to create a more diverse DDA for affective PX (Denisova et al., 2021) by developing the performance of the state prediction model and designing an elaborate score function.

8. Conclusion

Herein, we propose state-specific (CH, CO, VA, and FL) DDA agents that aim to improve PX by observing the player state. This was accomplished by a simple concept that uses player states as a score function to determine the action of game agents to automatically adjust difficulty, unlike manual modification on the heuristics. The proposed approach adjusted the AI opponents according to their target states and improved gameplay compared to a heuristic-based baseline (HP) agent. Despite being exposed to similar AI action strategies according to targeting player states, players experienced differing levels of gameplay satisfaction. We can improve DDA to satisfy various gameplay personalities. This approach has the potential for broad applications, expanding beyond AI-based fighting games and the gaming industry. For education and healthcare, this approach may be used to improve satisfaction with general application experiences in real-time.

Acknowledgments

This research was supported by the National Research Foundation of Korea (NRF) funded by the MSIT (2021R1A4A1030075).

Appendix. MCTS iterations

For MCTS, we used multi-threading to develop our DDA agent. We used 3700X - 8 core processors with 64 GB of memory machine for the experiment. The baseline agent and our agents iterated MCTS 900 to 1300 times within 165 ms in this environment. In conclusion, the baseline was slightly more iterated than ours on average, but there was no significant performance degradation for MCTS due to the evaluating model.

AGENT	MIN	MAX	AVG	STD
Baseline	913	1327	1158.22	113.25
CH	917	1311	1132.37	122.40
CO	900	1302	1114.70	97.28
VA	902	1297	1097.13	103.36
FL	900	1240	1021.97	95.88

Appendix. Pilot test study

Pilot test study result. 16 AIs leagued three rounds per opponent. Therefore, the total number of games in the league is 360. Among them, four games were tied. Five game research experts scored for each AI whether it looked like a human player.

AI	WIN	RANK	p1	p2	p3	p4	p5	MEAN	SD
ReiwaThunder	44	1	5	5	4	3	4	**4.2**	0.84
Thunder	37	2	3	4	3	4	3	3.4	0.55
LGIST_Bot	32	3	2	1	1	2	2	1.6	0.55
KotlinTestAgent	31	4	3	2	1	2	2	2	0.71
Dora	30	5	4	4	3	3	4	3.6	0.55
MctsAi	28	6	2	3	3	2	2	2.4	0.55
JayBot_GM	27	7	2	3	2	1	2	2	0.71
Toothless	26	8	3	2	4	1	1	2.2	1.3
FalzAI	24	9	4	4	4	3	4	3.8	0.45
MogakuMono	24	9	3	3	3	3	4	3.2	0.45
HaibuAI	21	11	4	5	5	4	5	**4.6**	0.55
SimpleAI	13	12	2	1	2	2	1	1.6	0.55
BCP	7	13	3	2	2	3	3	2.6	0.55
DiceAI	7	13	2	3	3	3	2	2.6	0.55
UtalFighter	3	15	2	3	3	2	2	2.4	0.55
TOVOR	2	16	4	4	4	5	4	**4.2**	0.45

Appendix. Parameter setting

Hyperparameter settings for each ML method. The names of the parameters are the names used in the scikit-learn.

ML Algorithm	Hyperparameters	Range
Logistic Regression	C	[-3 3]
	penalty	["l1", "l2"]
SVM	kernel	["rbf", "poly", "linear"]
	gamma	[0.001, 0.0001]
	C	[1, 10, 100, 1000]
LightGBM	n_estimators	[8, 24, 100]
	learning_rate	[0.01, 0.1]
	num_leaves	[6, 8, 12, 15, 31]
	reg_lambda	[0, 1, 2, 6]
	reg_alpha	[0, 1, 2, 6]
Random Forest	n_estimators	[100, 300, 400, 1300]
	max_depth	[None, 10, 15]
	max_features	['auto', 2, 3]
	min_samples_leaf	[1 6]
	min_samples_split	[1 6]

References

Afergan, D., Peck, E., Solovey, E., Jenkins, A., Hincks, S., Brown, E., Chang, R., & Jacob, R. (2014). Dynamic difficulty using brain metrics of workload. In *Proc. ACM CHI 2014 Human Factors in Computing Systems Conference*. doi:10.1145/2556288.2557230.

Ang, D., & Mitchell, A. (2017). Comparing effects of dynamic difficulty adjustment systems on video game experience. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play* (p. 317–327). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3116595.3116623.

Ang, D., & Mitchell, A. (2019). Representation and frequency of player choice in player-oriented dynamic difficulty adjustment systems. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play* CHI

PLAY '19 (p. 589–600). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3311350.3347165. doi:10.1145/3311350.3347165.

Auer, P. (2003). Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, *3*, 397–422.

Bonferroni, C. (1936). Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, *8*, 3–62.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32. URL: http://dx.doi.org/10.1023/A%3A1010933404324. doi:10.1023/A:1010933404324.

Chanel, G., Rebetez, C., Betrancourt, M., & Pun, T. (2011). Emotion assessment from physiological signals for adaptation of game difficulty. *IEEE Transactions on Systems, Man, and Cybernetics.A: Systems and Humans*, *41*, 1052–1063. URL: https://archive-ouverte.unige.ch/unige:47409. ID: unige:47409.

Chaslot, G., Bakkes, S., Szita, I., & Spronck, P. (2008). Monte-carlo tree search: A new framework for game ai. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* AIIDE'08 (p. 216–217). AAAI Press.

Chen, J. (2007). Flow in games (and everything else). *Commun. ACM*, *50*, 31–34. URL: https://doi.org/10.1145/1232743.1232769. doi:10.1145/1232743.1232769.

Constant, T., & Levieux, G. (2019). Dynamic difficulty adjustment impact on players' confidence. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (p. 1–12). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3290605.3300693.

Cowley, B., Charles, D., Black, M., & Hickey, R. (2008). Toward an understanding of flow in video games. *Comput. Entertain.*, *6*. URL: https://doi.org/10.1145/1371216.1371223. doi:10.1145/1371216.1371223.

Cronbach, L. (1951). Coefficient alpha and the internal structure of tests. *Psychometirka*, *16*, 297–334. URL: http://www.springerlink.com/content/n435u12541475367.

Csikszentmihalyi, M. (1991). *Flow: The Psychology of Optimal Experience*. New York, NY: Harper Perennial. URL: http://www.amazon.com/gp/product/0060920432/ref=si3_rdr_bb_product/104-4616565-4570345.

Demediuk, S., Tamassia, M., Raffe, W. L., Zambetta, F., Li, X., & Mueller, F. (2017). Monte carlo tree search based algorithms for dynamic difficulty adjustment. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)* (pp. 53–59). doi:10.1109/CIG.2017.8080415.

Denisova, A., Bopp, J. A., Nguyen, T. D., & Mekler, E. D. (2021). "whatever the emotional experience, it's up to them": Insights from designers of emotionally impactful games. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* CHI '21. New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3411764.3445286. doi:10.1145/3411764.3445286.

Denisova, A., & Cairns, P. (2015). Adaptation in digital games: The effect of challenge adjustment on player performance and experience. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play* CHI PLAY '15 (p. 97–101). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/2793107.2793141. doi:10.1145/2793107.2793141.

Duque, M. G., Palm, R. B., Ha, D., & Risi, S. (2020). Finding game levels with the right difficulty in a few trials through intelligent trial-and-error. *2020 IEEE Conference on Games (CoG)*, (pp. 503–510).

Fernandez B., H. D., Mikami, K., & Kondo, K. (2017). Adaptable game experience based on player's performance and eeg. In *2017 Nicograph International (NicoInt)* (pp. 1–8). Los Alamitos, CA, USA: IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/NICOInt.2017.11. doi:10.1109/NICOInt.2017.11.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, *32*, 675–701. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1937.10503522. doi:10.1080/01621459.1937.10503522. arXiv:https://www.tandfonline.com/doi/pdf/10.1080/01621459.1937.10503522.

Frommel, J., Fischbach, F., Rogers, K., & Weber, M. (2018). Emotion-based dynamic difficulty adjustment using parameterized difficulty and self-reports of emotion. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play* CHI PLAY '18 (p. 163–171). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3242671.3242682. doi:10.1145/3242671.3242682.

Hagelback, J., & Johansson, S. J. (2009). Measuring player experience on runtime dynamic difficulty scaling in an rts game. In *2009 IEEE Symposium on Computational Intelligence and Games* (pp. 46–52). doi:10.1109/CIG.2009.5286494.

Hunicke, R. (2005). The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology* ACE '05 (p. 429–433). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/1178477.1178573. doi:10.1145/1178477.1178573.

Hunicke, R., & Chapman, V. (2004). Ai for dynamic difficulty adjustment in games. In *Challenges in game artificial intelligence AAAI workshop*. volume 2.

IJsselsteijn, W., de Kort, Y., & Poels, K. (2013). *The Game Experience Questionnaire*. Technische Universiteit Eindhoven.

Ishihara, M., Ito, S., Ishii, R., Harada, T., & Thawonmas, R. (2018). Monte-carlo tree search for implementation of dynamic difficulty adjustment fighting game ais having believable behaviors. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)* (pp. 1–8). doi:10.1109/CIG.2018.8490376.

Ishii, R., Ito, S., Ishihara, M., Harada, T., & Thawonmas, R. (2018). Monte-carlo tree search implementation of fighting game ais having personas. (pp. 1–8).

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* NIPS'17 (p. 3149–3157). Red Hook, NY, USA: Curran Associates Inc.

Kivikangas, J. M., Chanel, G., Cowley, B., Ekman, I., Salminen, M., Järvelä, S., & Ravaja, N. (2011). A review of the use of psychophysiological methods in game research. *journal of gaming & virtual worlds*, *3*, 181–199.

Kneller, W., Higham, P., & Hobbs, M. (2012). Measuring manual dexterity and anxiety in divers using a novel task at 35–41 m. *Aviation, space, and environmental medicine*, *83*, 54–57.

Kuikkaniemi, K., Laitinen, T., Turpeinen, M., Saari, T., Kosunen, I., & Ravaja, N. (2010). The influence of implicit and explicit biofeedback in first-person shooter games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (p. 859–868). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/1753326.1753453.

Kusano, T., Liu, Y., Paliyawan, P., Thawonmas, R., & Harada, T. (2019). Motion gaming ai using time series forecasting and dynamic difficulty adjustment.

In *2019 IEEE Conference on Games (CoG)* (pp. 1–6). doi:10.1109/CIG.2019.8847991.

Liu, A.-J., Wu, T.-R., Wu, I.-C., Guei, H., & Wei, T.-H. (2020). Strength adjustment and assessment for mcts-based programs [research frontier]. *IEEE Computational Intelligence Magazine*, *15*, 60–73. doi:10.1109/MCI.2020.2998315.

Liu, C., Agrawal, P., Sarkar, N., & Chen, S. (2009). Dynamic difficulty adjustment in computer games through real-time anxiety-based affective feedback. *International Journal of Human–Computer Interaction*, *25*, 506–529. URL: https://doi.org/10.1080/10447310902963944. doi:10.1080/10447310902963944. arXiv:https://doi.org/10.1080/10447310902963944.

Lora, D., Sánchez-Ruiz, A., Gonzalez-Calero, P., & Gómez-Martín, M. (2016). Dynamic difficulty adjustment in tetris. In *Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference*.

Lu, F., Yamamoto, K., Nomura, L. H., Mizuno, S., Lee, Y., & Thawonmas, R. (2013). Fighting game artificial intelligence competition platform. In *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)* (pp. 320–323). doi:10.1109/GCCE.2013.6664844.

Moon, H.-S., & Seo, J. (2020). Dynamic difficulty adjustment via fast user adaptation. In *Adjunct Publication of the 33rd Annual ACM Symposium on User Interface Software and Technology* UIST '20 Adjunct (p. 13–15). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3379350.3418578. doi:10.1145/3379350.3418578.

Mourato, F., Birra, F., & dos Santos, M. P. (2014). Difficulty in action based challenges: Success prediction, players' strategies and profiling. In *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology* ACE '14. New York, NY, USA: Association for Com-

puting Machinery. URL: https://doi.org/10.1145/2663806.2663832. doi:10.1145/2663806.2663832.

Nacke, L. E., Grimshaw, M. N., & Lindley, C. A. (2010). More than a feeling: Measurement of sonic user experience and psychophysiology in a first-person shooter game. *Interacting with Computers*, *22*, 336–343. doi:10.1016/j.intcom.2010.04.005.

Oh, I., Rho, S., Moon, S., Son, S., Lee, H., & Chung, J. (2021). Creating pro-level ai for a real-time fighting game using deep reinforcement learning. *IEEE Transactions on Games*, (pp. 1–1). doi:10.1109/TG.2021.3049539.

Olesen, J. K., Yannakakis, G. N., & Hallam, J. (2008). Real-time challenge balance in an rts game using rtneat. In *2008 IEEE Symposium On Computational Intelligence and Games* (pp. 87–94). doi:10.1109/CIG.2008.5035625.

Parnandi, A., & Gutierrez-Osuna, R. (2015). A comparative study of game mechanics and control laws for an adaptive physiological game. *Journal on Multimodal User Interfaces*, *9*, 31–42. doi:10.1007/s12193-014-0159-y.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pfau, J., Smeddinck, J. D., & Malaka, R. (2020). Enemy within: Long-term motivation effects of deep player behavior models for dynamic difficulty adjustment. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (p. 1–10). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3313831.3376423.

Rani, P., Sarkar, N., & Liu, C. (2005). Maintaining optimal challenge in computer

games through real-time physiological feedback. *Foundations of Augmented Cognition*, *184*.

Sarkar, A., & Cooper, S. (2019). Transforming game difficulty curves using function composition. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (p. 1–7). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3290605.3300781.

Silva, M. P., do Nascimento Silva, V., & Chaimowicz, L. (2017). Dynamic difficulty adjustment on moba games. *Entertainment Computing*, *18*, 103–123. URL: https://www.sciencedirect.com/science/article/pii/S1875952116300350. doi:https://doi.org/10.1016/j.entcom.2016.10.002.

Stahlke, S., Nova, A., & Mirza-Babaei, P. (2020). Artificial players in the design process: Developing an automated testing tool for game level and world design. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play* CHI PLAY '20 (p. 267–280). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3410404.3414249. doi:10.1145/3410404.3414249.

Stein, A., Yotam, Y., Puzis, R., Shani, G., & Taieb-Maimon, M. (2018). Eeg-triggered dynamic difficulty adjustment for multiplayer games. *Entertain. Comput.*, *25*, 14–25.

Stephenson, M., & Renz, J. (2019). Agent-based adaptive level generation for dynamic difficulty adjustment in angry birds. *arXiv preprint arXiv:1902.02518*, .

Tognetti, S., Garbarino, M., Bonarini, A., & Matteucci, M. (2010). Modeling enjoyment preference from physiological responses in a car racing game. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games* (pp. 321–328). doi:10.1109/ITW.2010.5593337.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, *1*, 196–202.

Xue, S., Wu, M., Kolen, J., Aghdaie, N., & Zaman, K. A. (2017). Dynamic difficulty adjustment for maximized engagement in digital games. In *Proceedings of the 26th International Conference on World Wide Web Companion* WWW '17 Companion (p. 465–471). Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee. URL: https://doi.org/10.1145/3041021.3054170. doi:10.1145/3041021.3054170.

Zohaib, M., & Nakanishi, H. (2018). Dynamic difficulty adjustment (dda) in computer games: A review. *Adv. in Hum.-Comp. Int.*, *2018*. URL: https://doi.org/10.1155/2018/5681652. doi:10.1155/2018/5681652.