Identifying Playstyles in Games with NEAT and Clustering

Yu Iawasaki
iwasaki@mas.cs.tsukuba.ac.jp

Koji Hasebe
hasebe@cs.tsukuba.ac.jp

Department of Computer Science, University of Tsukuba
1-1-1, Tennodai, Tsukuba 305-8573, Japan

***Abstract*—In the research of agents playing computer games, attempts have been made to create agents with particular playstyles. A typical approach adopted in previous studies is to define the reward functions so that the agent has the desired playstyle. However, based on this approach, it is difficult to identify other playstyles that have not been specified in advance. The purpose of this paper is to create agents with playstyles by combining a genetic algorithm called NeuroEvolution of Augmenting Topologies (NEAT) and clustering based on the logs of gameplay by agents. Here, individuals of NEAT are classified in some species by clustering according to the play log and crossed with the same species. We evaluated our method by an experiment with a roguelike game and observed that multiple playstyles were identified. Furthermore, by comparing the proposed method with the method using only NEAT, it was shown that the bias in the number of individuals among playstyles can be suppressed.**

***Index Terms*—Game AI, Playstyle, NEAT, Clustering.**

I. Introduction

Studies on computer programs (agents) that play games are actively conducted. Also, numerous studies have been conducted to achieve other goals than the pure strength [1], for example, entertaining humans or imitating human plays. Recently, attempts have been made to create various types of agents with different playstyles [2]–[4]. Here, the playstyle means a set of characteristic actions of the player [5].

A typical approach adopted in previous studies is to define the reward functions beforehand so that the agent has the desired playstyle. However, based on this approach, only the style assumed in advance can be identified. Thus, as an application of this method, to use the created agents for test play, it is necessary to identify playstyles that are not expected by game designers and engineers.

To address this issue, the purpose of this paper is to identify agents with multiple playstyles without predefining styles. The basic idea is to combine an optimization method using a genetic algorithm called NeuroEvolution of Augmenting Topologies (NEAT) [6] with clustering related to the logs of games played by agents. In this method, NEAT searches for many individuals who highly adapt to the game (i.e., individuals who play the game well) by mutation and crossover. Furthermore, by clustering groups based on the individuals' play logs in each population, those who perform similarly are grouped into the same cluster. This allows us to search for multiple playstyles that are optimized for the game, while protecting against excessive natural selection by dividing them into species for similar playstyles.

In this study, we evaluated the proposed method through an experiment with MiniDungeons [2], which is a kind of roguelike game. We designed a simple stage of the game as well as a single reward function that determined the total score of the game. From the experiment, we observed that four playstyles were identified by the proposed method. Furthermore, a comparative experiment was conducted between the proposed method and a case where only NEAT was used. As a result, the bias in the number of individuals among different playstyles can be suppressed. This suggests that the use of clustering would help identify multiple playstyles in games.

Our proposed method is considered to have multiple applications. First, agents with playstyles will help to efficiently discover potential bugs in test play in game development. Also, games, where enemies with different playstyles appear, will make human players more fun.

This paper is organized as follows. Section II presents related work. Section III gives formal definitions of games and playstyles. Section IV describes our proposed method. Section V presents the experimental results. Finally, Section VI concludes the paper and presents future work.

II. Related Work

Tychsen et al. [5] is one of the earliest studies on playstyle in games. They defined the notion of playstyle as a set of distinct behaviors of players.

Holmgård et al. [2], [3] gave a model of the evolution of player's decision making and created five agents with different playstyles through learning. They created agents specified by linear networks and used an evolutionary strategy. They defined reward functions for these playstyles respectively.

Tampuu et al. [4] created two playstyles of cooperation and hostility on the hockey game called Pong of Atari 2600. These playstyles were implemented by defining two distinct reward functions based on Deep-Q-Network [7].

As discussed above, these studies have adopted the same approach that defines reward functions so that the agent has the intended playstyle. These agents are applied for level design, automated play testing [2], and for opponents who attract humans. However, because this method assumes the playstyle in advance, there is a possibility that biases affect due to game designers' and engineers' preconceptions. On the other hand, our method does not define any specific playstyles in advance thereby identifying unexpected playstyles.

III. Definition of Playstyle

In this section, we give a formal definition of the playstyle. First, we define games as complete information extensive-form games in game theory (cf. [8]). An extensive-form game is a situation in which one or more players make consecutive decisions according to a policy and each player obtains a reward according to the result. However, for the sake of simplicity, we only consider games with a single player.

Definition 1. A complete information extensive-form game is a tuple $(A, H, Z, \chi, \sigma, u)$, where:

- A is a set of actions;
- H is a set of non-terminal choice nodes of game tree;
- Z is a set of terminal nodes of game tree, disjoint from H;
- $\chi : H \to 2^A$ is the action function, which assigns to each choice node a set of possible actions;
- $\sigma : H \times A \to H \cup Z$ is the successor function, which maps a non-terminal node and an action to a new node;
- $u : Z \to \mathbb{R}$ is the reward (or, utility) function.

Here, $\mathbb{R}$ denotes the set of real numbers.

For a given game $G = (A, H, Z, \chi, \sigma, u)$, a strategy is defined as follows.

Definition 2. A strategy $s : H \to A$ is a function for a player to choose an action $a \in A$ at each node $h \in H$.

In the following, S is used to denote the set of strategies. Also, we use the word "play log" that means a log of gameplay by an agent, represented as a metric (n-tuple) of real numbers. For example, each dimension of a vector includes the number of collected coins or the number of killed enemies. The variable $y \in \mathbb{R}^n$ ($n \geq 1$) is used to indicate a play log and the symbol Y denotes the set of play logs.

Next, the play function is defined below.

Definition 3. A play function $p : S \to Y$ is a function that returns a play log for a given strategy s.

That is, play function determines the history of players' behaviors in games according to its strategy.

Next, the notion of a valid solution is defined as follows.

Definition 4. Let s be a strategy for a game G. Let z be a terminal node in G according to s. For a given $\theta \in \mathbb{R}$, s is a valid solution of G if $u(z) > \theta$. Here, θ is called a threshold of the valid solution.

The symbol S^* is used to denote the set of valid solutions. Finally, the playstyle is defined as follows.

Definition 5. Let S^* be the set of valid solution of a game G for a given threshold θ. A list $(S_1^*, \ldots, S_c^*)$ with $S_i^* \subset S^*$ (for $i = 1, \ldots, c$) is called a list of playstyles of G with granularity $c \geq 1$ if it satisfies the following conditions.

- $S_i^* \cap S_j^* = \emptyset \quad (\forall i, j \leq c)$;
- $\| \frac{1}{|S_i^*|} \sum_{s \in S_i^*} p(s) - \frac{1}{|S_j^*|} \sum_{s \in S_j^*} p(s) \| \geq \delta_b \quad (\forall i, j \leq c)$;
- $\forall s_a, s_b \in S_i^* \ (\|p(s_a) - p(s_b)\| \leq \delta_i) \quad (\forall i \leq c)$;

where $\delta_b \in \mathbb{R}$ is the threshold of distance between playstyles and $\delta_i \in \mathbb{R}$ is the threshold of distance between strategies of the playstyle itself.

IV. Playstyle Generation Method

In this section, we present our proposed method to generate agents with multiple playstyles, combining NEAT with a clustering technique. NEAT [6] is a genetic algorithm that optimizes the structures and weights in feedforward neural networks. Population evolves to get higher values of the fitness function with mutation, crossover, and selection. Notably, NEAT provides a mechanism called speciation. The speciation combines individuals with similar topologies as a group (spices) to protect them with new characteristics from natural selection. The index of species classification is the genetic distance $\delta(i, j)$, which represents the topological difference between individuals i and j. NEAT selects individuals solely based on the fitness and the topology similarity and does not consider the game log, which is used to characterize the playstyle. Therefore, individuals who are slightly inferior are eliminated, although they can be sufficiently recognized as individuals with different playstyles.

To solve this problem, we introduce an algorithm called C-NEAT obtained by incorporating a clustering method into NEAT. In each population of mating, clustering is performed based on the individuals' play log. Here, close species are mated so that the optimization is not aggregated in one direction by a single fitness function.

Algorithm 1 C-NEAT

```
1: function C-NEAT($T, k$)
2:     population, species, fitnesses ← init()
3:     for $t \leftarrow 1, T$ do
4:         population ← reproduce(population, species, fitnesses)
5:         fitnesses ← evaluate(population)
6:         cluster ← clustering($k$, population)
7:         species ← speciate(population, species, cluster)
8:     end for
9: end function
```

The detailed algorithm of C-NEAT is presented in Algorithm 1. In Line 5 each individual plays the game and sets its fitness based on the sum of the rewards obtained until the end condition. Here, the variables "population" and "fitnesses" indicate the arrays of individuals and fitness, respectively. At this step, each individual receives and stores the play log, which includes information such as the number of events and the history of actions. This play log is in the form of vectors or matrices.

In Line 6, individuals are classified into k clusters with clustering methods, such as k-means, based on these play logs. At this step, the variable "cluster" stores the correspondence between individuals and cluster IDs. The value of k is set as the number of different playstyles desired to be created.

In clustering, the function σ, which determines whether two different individuals i and j belong to the same cluster, are defined by

$$\sigma(i,j) = \begin{cases} 0 & (c_i = c_j) \\ 1 & (c_i \neq c_j), \end{cases}$$

where c_i indicates the cluster the individual i belongs.

Based on the result of clustering, in Line 7 speciation is performed to classify individuals into species. Here, the variable "species" determines the correspondence between individuals and species IDs. The distance of genomes $\delta'(i,j)$ in C-NEAT is defined by the following equation

$$\delta'(i,j) = \delta(i,j) + \alpha \cdot \sigma(i,j),$$

where α is a parameter. The bigger the value of α, the stronger correlation to the clustering result than the topological structure. This equation classifies individuals with similar play logs and topology into the same species as much as possible.

Finally, Line 4 produces new individuals with mutations and crossover between the same species. These operations from Line 4 to Line 7 are repeated T times to analyze the last population.

V. Evaluation by Experiments

A. Basic Settings

In this study, we conducted experiments using a roguelike game called MiniDungeons [2], [3], developed by Holmgård et al., to evaluate the performance of our proposed method. Multiple reward functions were defined in their work for each playstyle to emerge. In contrast, we designed a stage and defined a single reward function.

MiniDungeons is a game in which the player (agent) can repeatedly move to reach the exit. Agents can earn a positive reward for acquiring each treasure and potion or defeating monsters. When the agent fights a monster, he dies if his physical strength becomes zero, acquiring a negative reward. Furthermore, a negative reward is obtained as a penalty for the passage of time for each movement.

In our experiments, each individual in C-NEAT receives the state and reward from the environment, and outputs the selected action with the network. This operation is repeated until the agent reaches the goal, the physical strength is 0 or less, or the specified number of turns has passed. After the end of the play, the total rewards obtained during this play are set to the fitness value of the agent, and the agent's play log is recorded. Then, based on this play log, individual clustering was performed, and a new population was created by mutation and crossover. The above steps are repeated a fixed number of times.

The neurons of the input and output layers in the initial state of the network of the individual is presented in Table I. This setting is based on the study by Holmgård et al. [2]. At each action in the game, the individual receives information from the environment about the shortest distance to each object and the remaining health as inputs. Here, there are two possible distances: (a) the distance when approaching the object safely avoiding monsters (b) the distance without avoiding them. Therefore, a total of eight input neurons were provided. Similarly, there were seven output neurons. At each decision point, the action corresponding to the output neuron with the maximum value is selected, and the agent moves one square toward the object. These input and output neurons are initially fully connected, and neurons and edges are added or removed from the hidden layer during natural selection.

TABLE I: The neurons of the input and output layers in the initial state of the network.

Input layer
Health
Distance to Monster
(Safe) Distance to Treasure
(Safe) Distance to Potion
(Safe) Distance to Exit

Output layer
Move to Monster
(Safe) Move to Treasure
(Safe) Move to Potion
(Safe) Move to Exit

B. Parameter Settings

To evaluate the effectiveness of clustering on the identification of playstyles, we compared with the case where playstyles were identified only by NEAT without clustering. Here, to facilitate the evaluation, we designed a stage and rewards so that three possible optimal styles can be considered. Therefore, we set the parameters as $k = 3$, $\theta = -\infty$, $\delta_b = -\infty$, and $\delta_i = \infty$, respectively.

The stage used in the experiments is shown on the left of Figure 1. Furthermore, the rewards for each action and value of each game parameter are presented in Tables II and III, respectively. The stage and rewards were designed to have two optimal styles and one suboptimal style. The four playstyles including a non-optimal style assumed to occur in the final population are presented on the right of Figure 1. In this game, Style 1 collects treasures and Style 2 defeats the monsters to reach the goal (13 points). Style 3 goes straight to the exit (12 points) and Style 4 fights against monsters and die (-26 points). Except for these four playstyles, all other behaviors were counted as "others."

The other parameter settings are as follows. The nonhierarchical k-means method was adopted, where the number of clusters was set to 3. Before clustering, we conducted normalization of the play logs and principal component analysis (PCA). 20 trials were performed for averaging.

A play log was represented as a vector. This stored the number of times the agent visited each type of squares, and the number of times the action type of the output layer is selected in Table I.

C. Experimental Results

Figure 2 shows the results of classifying the agents in the final population based on the playstyles presented in Figure 1 for each case using NEAT and C-NEAT. In the case of NEAT, about 75% of agents followed Style 1 to collect treasures. This is because if the agent chooses the lower route first without taking the potions, its health will be 0 and it will

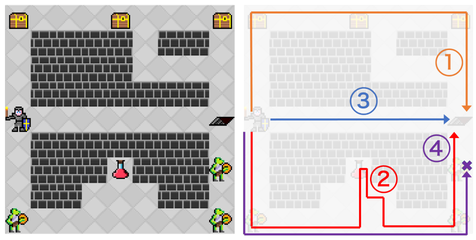

Fig. 1: Stage (Left) and the four possible behaviors (Right).

TABLE II: Reward setting.

Reward	Value
Movement	-1
Reaching the exit	20
Acquisition of a tresure	3
Acquisition of a potion	1
Defeating a monster	4
Death	-20

get a big negative reward. Thus, many agents with this style eventually survived by avoiding dangerous monsters and get the maximum score safely. However, when C-NEAT was used, Style 2 to defeat monsters accounted for 20.8%, which was about twice as much as when NEAT alone was used. This is because the clustering of play logs can protect agents with different styles from natural selection by a single reward function. The same was observed for other styles. From the above results, it was shown that the population of playstyles by NEAT is useful and that the use of C-NEAT can suppress evolution in a single direction. Since this experiment used only a simple game, further evaluation is needed to confirm its usefulness for more complex games with potentially more styles.

The result of PCA of the play logs of the final population in C-NEAT is shown in Figure 3. The coordinates of each point in this figure represent the play log of each of the 200 individuals. Also, the color and shape indicate the playstyles of an individual. The play logs of individuals with the same playstyle are arranged in a close space, as shown in this figure. Therefore, it is considered that the species of different playstyles can be promoted based on the play log by properly determining the number of clusters.

VI. Conclusions and Future Work

In this study, we proposed a method for identifying different playstyles only by a single reward function in a game without predefining specific playstyles and reward functions. The basic idea was to combine NEAT, with the clustering of play logs. Specifically, individuals adapted to the game as they evolved through natural selection of NEAT. Clustering of play logs suppressed monotonous evolution so that more playstyles survived. The experiment using a roguelike game shows that our method can generate agents with multiple playstyles. We also observed that by using clustering, it is possible to suppress individual bias between playstyles.

However, to make the analysis easier, the experiments in this study use only a simple game enough to predict the playstyles that are generated. Therefore, further evaluation should clarify the effectiveness of our method for more complex games potentially including numerous playstyles. To increase the scalability of our method, we should improve the clustering and evolution mechanism to survive the sub-optimal playstyles.

TABLE III: Parameter settings.

Parameter	Value
Initial Health	30
Health required to defeat a monster	10
Health recovered by a potion	10
Maximum number of movements	80
Number of individuals in a population	200
Number of populations	200
Number of clusters	3

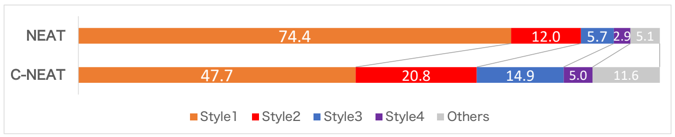

Fig. 2: Comparison of the ratios of generated playstyles.

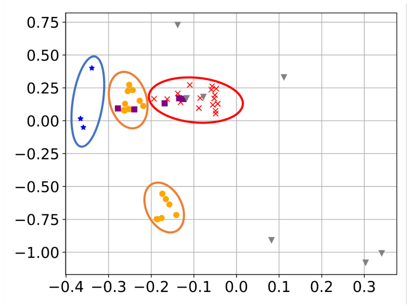

Fig. 3: Result of PCA of the play logs of the final population in C-NEAT.

References

[1] R. Lara-Cabrera, M. Nogueira-Collazo, C. Cotta, and A. Fernández-Leiva, "Game artificial intelligence: Challenges for the scientific community," *CEUR Workshop Proceedings*, vol. 1394, pp. 1–12, 01 2015.

[2] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Evolving personas for player decision modeling," *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8, 2014.

[3] ——, "Evolving models of player decision making: Personas versus clones," *Entertainment Computing*, vol. 16, pp. 95–104, 2016.

[4] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, 2017.

[5] A. Tychsen and A. Canossa, "Defining personas in games using metrics," in *Proceedings of the 2008 Conference on Future Play*, 2008, p. 73–80.

[6] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, 2002.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *ArXiv*, vol. 1312.5602, 2013.

[8] K. Leyton-Brown and Y. Shoham, "Essentials of game theory: A concise multidisciplinary introduction," *Synthesis lectures on artificial intelligence and machine learning*, vol. 2, no. 1, pp. 1–88, 2008.