Estimation of Player's Preference for Cooperative RPGs Using Multi-Strategy Monte-Carlo Method

Naoyuki SATO
Japan Advanced Institute of Science and Technology
Ishikawa, Japan
Email: satonao@jaist.ac.jp

Kokolo IKEDA
Japan Advanced Institute of Science and Technology
Ishikawa, Japan
Email: kokolo@jaist.ac.jp

Takayuki WADA
Japan Advanced Institute of Science and Technology
Ishikawa, Japan

***Abstract*—In many video games such as role playing games (RPGs) or sports games, computer players act not only as the opponents of the human player but also as team-mates. But computer players as team-mates often behave in a way that human players do not expect, and such mismatches cause bigger dissatisfaction than in the case of computer players as opponents.**

One of the reasons for such mismatches is that there are several types of sub-goals or play-styles in these games and the AI players act without understanding the human player's preference about them. The purpose of this study is to propose a method for developing computer team-mate players that estimate the sub-goal preferences of the team-mate human player and act according to these preferences.

For this purpose, we modeled the preferences of sub-goals as a function and decided the most likely parameters by a multi-strategy Monte-Carlo method, by referring to the past actions selected by the team-mate human player.

Additionally, we evaluated the proposed method through two series of experiments, one by using artificial players with various sub-goal preferences and another one by using human players. The experiments showed that the proposed method can estimate their preferences after a few games, and can decrease the dissatisfaction of human players.

I. Introduction

A simple and ultimate goal of computer intelligence for games is to entertain human players. To reach this goal, strength of computer game players have been investigated first, and a lot of methods have been developed. Currently computer players of many two-player board games such as Chess or Go are stronger than almost all human players. In the case of games where several players play in a team such as football, many approaches have also been proposed to make strong computer team members [1]. Such strong computer players may be good enough *opponents* to human players.

However, there are still few researches for making good computer *team-mates*. In many video games such as role playing games (RPGs) or sports games, computer players are needed not only as opponents of the human players but also as team-mates. Frequently, computer players as team-mates behave in a way that the human player does not expect, and such mismatches cause bigger dissatisfaction than in the case where computers are the opponents.

One of the reasons for such mismatches is that such games has not only the main goal of "winning" but also several "sub-goals", then the best action for winning is not necessarily the best action for satisfying the human player. We believe that to be a good team-mate, computer players should understand which sub-goals are preferred by the player by referring to the past actions selected by the player, and act according to that preference.

Our purpose is to implement team-mate computer players which can estimate human players' preference and decrease their dissatisfaction. For this purpose, preference of sub-goals is modeled by a function, and the most likely parameters are decided by a multi-strategy Monte Carlo method.

The structure of this paper is as follows. We introduce some related works in Section II and show an overview of the proposed method in Section III. The general algorithm is shown in Section IV. The target game is described in Section V, and then Section VI and VII describe respectively the preference function and the strategies specific to the game. Two series of experiments are done and shown in Section VIII and IX, and Section X concludes this paper.

II. Related Works

Compared to conventional researches which aim to make strong computer players in two-player board games, this paper deals with a different situation, that is (1) a behavior satisfactory to the player is pursued, (2) target games are role playing games (3) team playing is needed, and then (4) player modeling is needed.

Since a natural behavior of computer players is an important factor for player satisfaction, many researches have been done, and many competitions have been held [3]. For example, Fujii *et al.* proposed a method to produce human-like behaviors by introducing biological constraints in the search and learning algorithms, and showed its effectiveness on an action game Infinite Mario Bros. [2]. Believability is a similar but more complex concept than naturalness, Bernacchia *et al.* suggested that consistency based on character's purpose or personality is very important for believability [5].

Team playing is needed in multi-player games such as football, and many approaches have been tried for making strong computer teams. For example, Bakkes *et al.* proposed an evolutionary approach which can adapt the team strategy dynamically to the opponent team, and its ability to exploit the opponent patterns was shown in a shooting game Quake III [1].

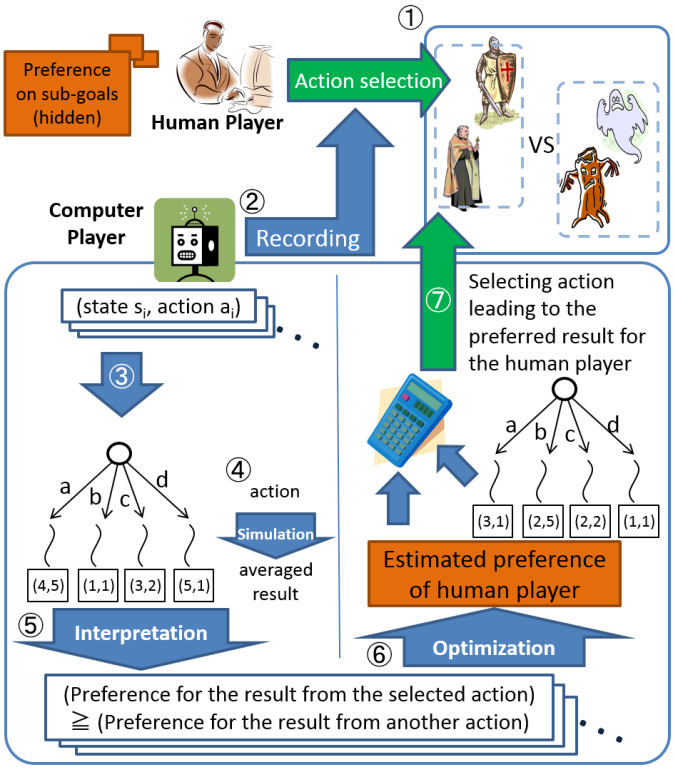

Fig. 1. Approach overview

Our target game is also a kind of multi-player games, called command-based role playing video games (RPGs), such as Wizardry series or Final Fantasy series. In such games, the player team is usually stronger than the opponent (monster) team, but the player team must fight against many teams in a row, and its status at the end of a battle is kept for the start of the next one. Then, players seek not only to just "win" but also to reach a "desirable win", for example they try to avoid injuries to their characters, to spend less magic powers, or to avoid a loss of time. We call such elements for a desirable win the sub-goals. It must be noted that the preferences on such sub-goals strongly depend on each human player, for example some players may prefer speedy but risky battles, while others may prefer safe but slow battle. Hence, there is a need to estimate such preferences.

While Bakkes's approach tried to exploit the tendency of *opponents*, we try in this research to estimate the preference of a *team-mate*, by referring to the past actions selected by the team-mate human player, and then we try to adjust the actions of computer team-mates to such preferences. Modeling of human player's behavior by referring to the selected actions itself is popular also in board games, especially for making strong players by referring to some professional game records. It is well known that $\alpha\beta$ tree search or Monte-Carlo tree search can be enhanced by such modeled action evaluation functions [14][16][6]. Also, modeling actions of the current opponent for exploitation have been widely tried [8][9][10].

Modeling of preferences on states (game/board situations) is also popular. Hoki *et al.* proposed a sophisticated method for learning state evaluation functions from game records [11], or Namai *et al.* tried to produce the play style of a specific player by using his game records [15]. We also try modeling preferences of each player, but two difficulties should be considered, *i.e.* there are many sub-goals and the state transition is stochastic. In the field of reinforcement learning, inverse reinforcement learning methods proposed by Y. Ng *et al.* try to suggest reward functions for agents by analyzing their policies [12].

For dealing with human complex preference on stochastic events, the concept of "utility" is frequently used [13]. For example, someone could reasonably want to do a gamble where he wins 100 dollars at 70% probability and loses 100 dollars otherwise, but he would probably refuse to do a similar gamble where he wins 100,000 dollars at 75% probability and loses 100,000 dollars otherwise. Such cases cannot be well explained by considering only plain average rewards, but can be well explained by the utility theory. Winning 100 dollars and losing 100 dollars have roughly the same absolute impacts for almost all people, but losing 100,000 dollars has a much bigger (negative) impact compared to winning 100,000 dollars. Such non-linear impacts can be described by using utility functions, and the "average utility" can be compared. Multiple features of an event or a matter, for example {price, speed, cornering, toughness, fuel efficiency} for a car, can also be included in the utility function. In this paper, we propose a method to learn and use a utility function of a human player in order to cooperate with him as a team-mate.

III. Approach

The goal of this research is to make an entertaining computer player for RPGs who can cooperate with a human player as a team-mate. In such games, computer players often behave in a way that the human player does not expect, and such mismatches cause big dissatisfaction.

There are many possible reasons of such mismatches, for example the best action for winning may be difficult to find in complex games, or the human player can misunderstand the situation even though the computer player selects the best action. But we believe that the main reason of such mismatches is the existence of sub-goals. For example, if the computer player selects the most probable action for winning, it is reasonable in a sense, but the human player may feel that the action is too slow or too magic-power consuming, and may prefer more speedy or magic-power saving action, even if it is a bit risky. So, estimation of preferences and adaptation to them are needed.

Figure 1 shows the overview of the proposed method. In this section, each procedure is briefly described in the order of the numbers on the figure.

1) **Target games**: In this research we focus on multi-player games where two teams battle against each other, and each team is composed of several characters. We consider games with discrete time-step and discrete action space, mainly command-based RPGs. The main goal of the players is to win each battle, but in addition, players try to achieve a "desirable win", in other words there are some sub-goals. The definition of a "desirable win" is unclear and depends on each player.
2) **Recording states and actions**: We assume one character is controlled by a human player. Each action of the

character is recorded, in a pair with the state (situation) in which the action is selected.

3) **Starting estimation**: It is impossible to estimate the preference of a player only by one selected action. The estimation is started after storing several pairs of states and actions. The estimation can be updated and used at any time through the battles.
4) **Calculation of averaged results**: In each state, the human player selects an action from some candidates, knowing that each different action will lead the team to a different result. Such expected result is estimated for each possible action, using Monte-Carlo simulations. Many simulations are done for each possible action in the given state, and the *averaged result* is calculated for each.
5) **Interpretation**: It is reasonable to consider that each human player selects the action which will lead the team to the *best* result according to his own preference. In other words, we can interpret the fact that "an action was selected" as "the averaged result of the selected action was more desirable than that of any other action.".
6) **Preference estimation**: We assume that each human player implicitly has his own preference function, which we model as a parameterized function that takes the averaged results of the Monte-Carlo simulations as inputs and returns a preference value. The parameters of this preference function are optimized so that the conditions interpreted in 5) are satisfied as well as possible.
7) **Action selection**: After estimating the preference function, the computer team-mate computes the averaged results of each possible action, computes the preference values, and selects the best desirable action.

In this way, desirable actions of team-mate computer player are suggested from observing only the humnan player's actions. Of course, there are cases in which a human player's preferences to actions of his team-mate and of his own, seem different (e.g. the human player choose attack actions eagerly to obtain some rewards by killing enemies while he want his team-mates to hold off from killing). However, we can suggest preferable team-mate's actions even in those situations by interpreting the simulated results precisely enough (e.g. we can focus on the killing player in each simulation results and estimate the preference to team-mates' holding off from killing).

IV. Algorithm

In this section, the whole algorithm is shown according to the stream of our approach (Figure 1). The notations of symbols are summarized in Table I.

A. Recording states and actions

In this research we employ Markov Decision Process (MDP) [7] as the model of target games, because almost all command-based RPGs can be modeled as a MDP. Let S be the discrete state space, and A the discrete action space. When a computer player needs to estimate the preference of a human player, the actions selected by the human player are recorded. The j-th state for the player is noted by s^j, the possible actions at that time are noted by $A_{s^j} \subset A$, and the selected action is noted by $a^j \in A_{s^j}$. The recorded information is a set of pairs of such states and actions, noted by $\{(s^j, a^j)\}^j$.

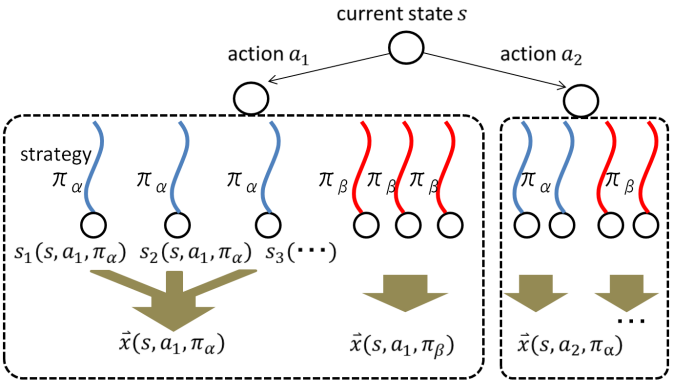

Fig. 2. Calculation of averaged results

B. Calculation of averaged results

To know the averaged results of each action, we use Monte-Carlo simulations. Each simulations is run from the state obtained just after the evaluated action is executed, until the end of the battle, using an action selection strategy $\pi : S \to \mathbb{R}^{|A|}$ which decides the selection probability of each possible action. We note the result (state) of the i-th simulation for a state s and an action a as $s_i(s, a, \pi)$.

Since a state includes too many values, an n-dimensional feature vector $\vec{x}_i(s, a, \pi)$ is extracted by using a function $S \to \mathbb{R}^n$. After the extraction, the averaged result of m simulations is calculated as follows:

$$\vec{x}(s, a, \pi) = \frac{1}{m} \sum_{i=1}^{m} \vec{x}_i(s, a, \pi) \tag{1}$$

Instead of random simulations where all legal actions are selected at the same probability, biased simulations are often employed to improve the performance of Monte-Carlo methods [16]. In such biased simulations, "good actions" are selected at higher probabilities. In the case of Chess or the game of Go, it is relatively easy to define "good actions" and so to employ a biased *single* simulation strategy.

However, in the case of RPGs, many sub-goals exist, then the "good actions" depend on the preference of each player. Therefore, we employ *multiple* simulation strategies, and calculate multiple averaged results $\vec{x}(s, a, \pi)$ for each strategy π respectively. The procedure is shown in Figure 2.

This procedure for calculating averaged results is used in this paper not only for estimating preference, but also for selecting the actual action.

C. Preference estimation

Being inspired by the utility theory, we assume that each human player has his own preference function and selects the best action for maximizing it. Since the preference function is hidden, it is needed to employ a parameterized function model and optimize the parameters.

As there are many candidates for such function $u : \vec{x} \to \mathbb{R}$, in this paper we employ a simple linear-sum model as follows:

$$u(\vec{x}(s, a, \pi), \vec{w}) = \vec{x}(s, a, \pi) \cdot \vec{w} \tag{2}$$

TABLE I. NOTATIONS OF SYMBOLS

$s \in S$	current state
$A_s \subset A$	possible actions at state s
$a \in A_s$	a possible action
$a^* \in A_s$	action selected by the human player
$\pi : S \to \mathbb{R}^{\|A\|} \in \Pi$	a simulation strategy
n	number of features
m	number of simulations
$s_i(s, a, \pi)$	the result (state) of i-th simulation using strategy π, from state s and action a
$\vec{x}_i(s, a, \pi) \in \mathbb{R}^n$	feature vector of state $s_i(s, a, \pi)$
$\vec{x}(s, a, \pi) \in \mathbb{R}^n$	average of $\{\vec{x}_i(s, a, \pi)\}_i$
$\vec{w}$	parameter vector of preference function
$u(\vec{x}, \vec{w}) \in \mathbb{R}$	preference value of $\vec{x}$ when using $\vec{w}$

Algorithm 1 Optimization of parameter $\vec{w}$ for preference function

```
for each w⃗ ∈ W do
  p_w⃗ = 0
end for
for each (s, a*) ∈ {(s^j, a^j)}^j do
  for each w⃗ ∈ W do
    u* = max_{π∈Π} u(x⃗(s, a*, π), w⃗)
    for each a ∈ A_s \ a* do
      if u* < max_{π∈Π} u(x⃗(s, a, π), w⃗) then
        p_w⃗ += 1
      end if
    end for
  end for
end for
return arg min_{w⃗∈W} p_w⃗
```

$\vec{x}(s, a, \pi) \in \mathbb{R}^n$ is an averaged result, and $\vec{w} \in \mathbb{R}^n$ is a parameter vector.

We can interpret the fact that "an action a^* was selected" as "the averaged result of a^* was more desirable than that of any other action". Considering that the simulation strategy π can be selected from a possible set Π, we can expect the following inequality to hold:

$$\max_{\pi \in \Pi} u(\vec{x}(s, a^*, \pi), \vec{w}) \geq \max_{\pi \in \Pi, \overline{a} \in A_s} u(\vec{x}(s, \overline{a}, \pi), \vec{w}) \quad (3)$$

In other words, if this inequality does not hold, it is probable that the preference function, and then also the parameters $\vec{w}$, are inadequate. Then, we try to find the best parameters which minimize the number of violations of the inequality.

At first, the possible parameters are limited to a finite set W. When an action a^* is observed, each candidate $\vec{w} \in W$ is tested, and if the inequality (3) does not hold, a penalty $p_{\vec{w}}$ for $\vec{w}$ is increased. Finally, the parameter with the minimum penalty $p_{\vec{w}}$ is selected and considered to be the most adequate parameter (See algorithm 1). If there are several vectors which have the minimum penalty, the average vector is adopted.

The gradient descent method is another feasible approach for this estimation. We think the gradient descent method works better in case of using feature vectors in higher dimensions. Because, our discrete space approach increases computational cost exponentially as the dimension grows.

D. Action selection

How to calculate the averaged results $\vec{x}(s, a, \pi)$ and how to estimate the preference function $u(\vec{x}, \vec{w})$, have been already described. In order to cooperate with the human player by selecting the action that he is expecting, the action that maximizes the preference, $\arg\max_{a \in A_s, \pi \in \Pi} u(\vec{x}(s, a, \pi) \cdot \vec{w})$, is selected. We call such player "MC player" in this paper.

V. GAME SETTINGS

The proposed algorithm is evaluated by using a command-based RPG designed for academic research so that the result is reproducible. This game is modeled by a MDP, and we describe in the following subsections its state space, action space and transition rules.

A. State: parameters of characters

This game consists of one battle between two teams, and each team consists of several characters. There are several parameters for each character, which are all observable from each other. In usual commercial RPGs, such information of the opponent team is hidden or incomplete, but it is possible to estimate them by repeating battles. In this paper, this state estimation phase is skipped in order to focus on the problem of the preference estimation.

- **Vitality (HP)**: HP of a character is decreased by the opponent's attack. If HP becomes below 0, the character is beaten. Variable parameter.
- **Maximum Vitality (MHP)**: HP can be increased by some skills, but it is limited by MHP. Constant parameter.
- **Magic Power (MP)**: MP is necessary for using some powerful skills, decreased by using them. Variable parameter.
- **Offensive Power (ATK)**: Character with higher ATK can give more damage to the opponent's HP. Constant parameter.
- **Defensive Power (DEF)**: Character with higher DEF will take less damage from the opponent's attack. Constant parameter.

B. Action

Each character selects a skill and its target, for example "single attack to enemy-1" or "greater heal to team-mate-2". The set of possible skills differ between the characters. The list of skills is as follows:

- **Single attack**: one opponent is selected, and he receives (attacker's ATK - opponent's DEF) damage to his HP.
- **Group attack**: one group of enemies is selected, and each of them receive 30 damage to their HP. MP of the user is decreased by 8.
- **Lesser heal**: one team-mate (or oneself) is selected, and target's HP is increased by 42, at most to MHP. Instead, MP of the user is decreased by 4.

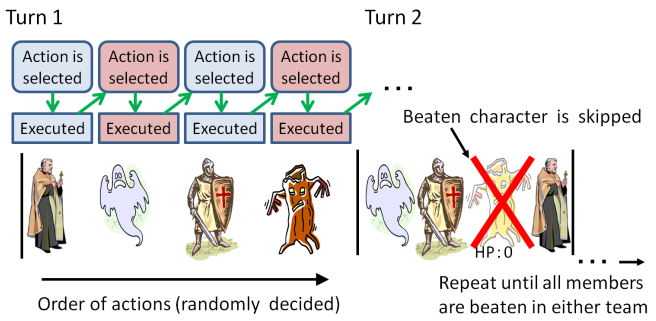

Fig. 3. Progress of a game. Turn-based with random order of the actions.

- **Greater heal**: the amount of healing and spent MP are bigger than lesser healing, 88 and 8 respectively.
- **Group heal**: all HPs of team-mates are increased by 160 at most to their MHPs, instead, MP of the user is decreased by 18.
- **Defense**: the damages taken are decreased by 50%, until the next action of the character is selected.

C. State transition

The overview of the state transitions is shown in Figure 3. Every alive characters act once in a *turn*. The order of actions is randomly decided, and the players cannot know the order.

In many games such as Wizardry series or Dragon Quest series, the actions of the human team are selected at the beginning of each turn. But in such case more complex decision making such as Nash Equilibrium is needed. For now, a simpler case is considered as the first step. The characters select their action not at the beginning of the turn, but when it is their time to act, and their action is executed soon after being selected.

After each selection of an action, a state transition is executed and the control is given to the next character. This loop is repeated until all characters of either team are beaten.

D. Settings

Compared to conventional two-player board games such as Chess, in RPGs usually two teams are completely asymmetric, and the parameters are not fixed but can vary widely. Then, we prepared five settings of battles, from easy ones to a difficult one corresponding to a "boss battle".

Table II shows the parameters of setting-2, in this setting there is no group including multiple enemies. It is easy to win in this setting, but it is a bit difficult to preserve MP because there is an enemy who can heal. In such case, sometimes players should select MP spending skill for preserving MP. It sounds paradoxical, this is a case where it is difficult for computers to learn the human player's preference.

VI. Feature vector and weight vector

In the proposed algorithm, an n-dimensional feature vector $\vec{x}$ is extracted from the result (state) of each simulation, and then an n-dimensional weight vector $\vec{w}$ is used to evaluate it. In this section, we describe the features used in this paper, and how the weights affect the computer player's behavior.

TABLE II. Setting-2, parameters and available skills of 5 characters

Character	group	HP	MP	ATK	DEF	Available skills
Hero-1		134	30	60	28	single attack, lesser heal, defense
Hero-2		108	60	44	34	single attack, group attack, lesser heal, greater heal, group heal, defense
Enemy-1	1	52	0	40	26	single attack
Enemy-2	2	82	32	38	32	single attack, lesser heal
Enemy-3	3	70	0	50	30	single attack

A. Feature vector

While the result after a simulation includes many values, only three features are temporarily extracted and used in this paper. Richer features may be required to represent human preference accurately, especially in the case of more complex RPGs, but it should be noted that higher dimensional optimizations require a bigger optimization cost.

The feature vector employed in this paper is as follows:

$$\vec{x} = (x_{HP}, x_{MP}, x_{Turn}) \tag{4}$$

Each feature element is calculated as follows:

$$x_{HP} = \frac{\text{averaged HP of player's team members}}{\text{averaged MHP of player's team members}} \tag{5}$$

$$x_{MP} = \frac{\text{averaged MP of player's team members}}{\text{averaged initial MP of player's team members}} \tag{6}$$

$$x_{Turn} = b - (\text{number of elapsed turns}) \times a \tag{7}$$

a and b are constant values for normalization. When the player's team wins with less final damages, a bigger x_{HP} is achieved. When the player's team wins after spending less MPs, a bigger x_{MP} is achieved. And when the player's team wins within less turns, a bigger x_{Turn} is achieved. It must be noted that the main goal of each battle, i.e. "winning", is embedded in x_{HP}, because x_{HP} has the lowest possible value (0) if the battle is lost.

B. Weight vector

In this setting, since the feature vector $\vec{x}$ is 3-dimensional, the weight vector $\vec{w}$ is also 3-dimensional. The elements of $\vec{w}$ represent the weight for x_{HP}, x_{MP}, x_{Turn}, respectively. Since a linear weighting model is employed in this paper, the weight for x_{HP} can be fixed to 1. For example, a weight vector $(1, 10, 0.1)$ represents a preference for preserving MP, and a weight vector $(1, 0.1, 0.1)$ represents a preference for avoiding damages.

For the proposed algorithm, a candidate set of parameters W should be prepared. We consider a two-dimensional 31×31 logscale matrix with x_{MP} and x_{Turn} as the axis, and where the minimum value and maximum value are set to $\frac{1}{32}$ and 32 respectively. In other words, W consists of a set of 961 candidate weight vectors.

TABLE III. BATTLE RESULTS WHEN USING THREE TYPICAL WEIGHTS

strategy	weight vector	HP (sum)	MP (sum)	Turn
multi	HP-preserving: $(\mathbf{1}, 0.1, 0.1)$	**238**	41	8.8
multi	MP-preserving: $(1, \mathbf{10}, 1)$	150	**80**	11.5
multi	Turn-oriented: $(1, 1, \mathbf{10})$	160	56	**6.2**
single	MP-preserving: $(1, \mathbf{10}, 1)$	159	63	6.9

C. *Effect of preference*

In this subsection, we investigate how weights affect the computer player's behavior. It is expected that different weights produce different behaviors, so we want to check that for example MP-oriented weights really produce an MP preserving behavior.

Here a "matching rate" is calculated to compare two players, by the following procedure.

1) Player-1 plays several games as a character (such as hero-1), k pairs of states and actions $\{(s_i, a_i)\}$ are recorded.
2) Each state s_i is also given to Player-2 playing the same character, and the chosen actions a'_i are recorded.
3) The number of matches between a_i and a'_i in the k states is counted, and the rate of matches is what we call the "matching rate".

It should be noted that the matching rate can be under 1.0 even if Player-1 and Player-2 are exactly the same, because randomness is used in the Monte-Carlo method algorithm.

Figure 4 shows the matching rates for varied $\vec{w} \in W$, compared to a fixed MC player using $\vec{w} = (1, \frac{1}{8}, \frac{1}{16})$. Battle setting-2 was used, hero-2 is controlled by the fixed MC player with these weights, and other characters were controlled by a fixed rule. Also Figure 5 shows the matching rates when comparing to another fixed MC player using $\vec{w} = (1, 4, 8)$. Please note that the directions of axes are opposite in the two figures, for the sake of visibility of the landscape.

In the case of the comparison with the fixed player using $\vec{w} = (1, \frac{1}{8}, \frac{1}{16})$, the highest matching ratio is achieved when $\vec{w} = (1, \frac{1}{8}, \frac{1}{16})$ itself, but there is a hill around it where the matching rate is over 70%. On the other hand, in the case of comparing with the fixed player using $\vec{w} = (1, 4, 8)$, there is a sharp ridge, $(1, 4, 8)$, $(1, 8, 16)$ or $(1, 16, 32)$ have good matching rates but $(1, 8, 8)$ or $(1, 4, 16)$ have significantly worse matching rates. It shows that different weight vectors can produce significantly close behaviors, so the accuracy of the preference estimation should be evaluated through this matching ratio of the behavior, instead of the values of the learned vector itself.

Next, to confirm if a weight for achieving something such as preserving MP really achieves it, simple experiments are done. Three typical weight vectors are prepared, and hero-2 in setting-2 is controlled by an MC player with one of them. 1000 battles were done, averaged results are summarized in Table III. It confirms that at least in this case, HP-preserving vector really achieved the best (highest) result about HP, MP-preserving vector really achieved the best (highest) result about MP, and Turn-oriented vector really achieved the best (fastest) result about Turn.

VII. MULTI-STRATEGY MONTE-CARLO

It is an important characteristic of the proposed method that several simulation strategies are used. Compared with a single-strategy (random-strategy) Monte-Carlo method, the multi-strategy Monte-Carlo method will allocate less number of simulations for each strategy but make the simulations more realistic. In this section, we explain the seven employed strategies, and the effect is shown through some experiments.

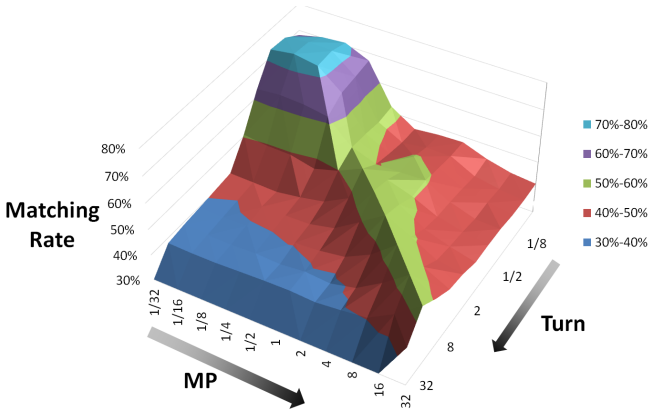

Fig. 4. Matching rates to a player using $(1, \frac{1}{8}, \frac{1}{16})$, logscale.

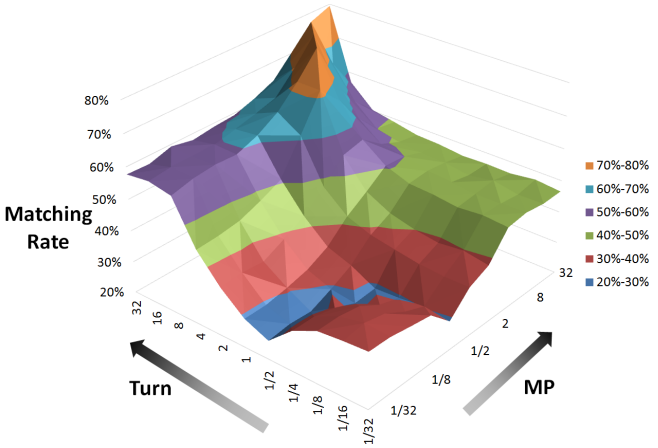

Fig. 5. Matching rates to a player using $w = (1, 4, 8)$, logscale.

A. *Employed strategies*

Several strategies should be prepared according to the target game. The best combination of an action and a strategy is finally selected by a Monte-Carlo method, then it is not necessary to prepare the candidate set so carefully. A foolish strategy can be included, because it will not be selected finally by the algorithm. Of course, considering the computational cost, foolish strategies should be removed if possible.

The candidate set used in this paper is as follows:

1) Random: all actions have the same selection probability.
2) Timely heal: the selection probability of healing skills is decreased when HPs of the team are not low. Usually, healing is a bad action (MP-spending and slow) if both team-mates are safe.
3) Offensive: attack skills are selected with a 5 times higher probability than other actions.
4) Single attacking: single attack is selected with a 5 times higher probability than other actions. This strategy is especially effective for MP-preserving purpose.

TABLE IV. WINNING RATES, MULTI-STRATEGY VS SINGLE-STRATEGY, HP-PRESERVING VS MP/TURN ORIENTED

weight vector	multi-strategy	single-strategy
HP-preserving: $(1, 0.1, 0.1)$	98.8%	96.0%
MP/Turn-oriented: $(1, 10, 10)$	83.6%	70.2%

5) Group attacking: group attack is selected with a 5 times higher probability than other actions. This strategy is especially effective for Turn-oriented purpose.
6) Timely heal + Single attacking
7) Timely heal + Group attacking

B. Effect of multi-strategy

It is well known in board games that using good simulations is effective for strong playing [16]. Here we show briefly that using multiple strategies is effective not only for strong playing, but also to obtain characteristic behaviors.

Firstly, the winning rates of two cases were compared by using battle setting-5 shown in Table V, assuming a boss-battle. In one case hero-2 was controlled by a single-strategy MC player, and in another case by a multi-strategy MC player. Characters except hero-2 were controlled by a fixed rule. Two preference weight vectors were also compared. Table IV shows the percentage (in 1000 games each) when both hero-1 and hero-2 were alive at the end of the battle. If we consider the winning performance, it is clear that HP-preserving (safer) preference can achieve better result, and that the multi-strategy MC player plays better than the single-strategy MC player.

Secondly, we compared the single-strategy and the multi-strategy to see whether characteristic behaviors according to the preferences were generated or not. The single-strategy MC player with MP-preserving preference $(1, 10, 1)$ was tested as shown in the last section, the result is also shown in Table III. Compared to the case of the multi-strategy MC player with MP-preserving preference, the remaining MP was significantly lower, and instead the number of elapsed turns was significantly fewer.

In particular, the single-strategy MC player often use group-attack, especially in the early stage of each battle. Because MP-consuming skills are often used in completely random simulations, then the small difference (of MP) caused by the first action can be easily cancelled by good/bad luck. On the other hand, in the case of the multi-strategy MC player, some of the prepared strategies (such as Timely-heal + Single attacking) can preserve MP also in simulations, then the small difference caused by the first action can be detected.

It is reasonable to consider that such advantages of a multi-strategy MC also contribute to the accuracy of preference estimation. Through some preliminary experiments, we observed that the accuracy when using a multi-strategy MC is about 10 points better (at most) than when using a single-strategy MC.

VIII. EVALUATION EXPERIMENTS USING ARTIFICIAL PLAYERS

The proposed preference estimation algorithm was evaluated through two series of experiments. In this section, the first series is shown. Artificial players were employed as the target players to be estimated, so that we can compare the preference of the target players and the estimated preference, and so that we can conduct many and various experiments easily. We employed 4 weight vectors (1, 0.071, 0.071), (1, 0.143, 18), (1, 12, 0.167) and (1, 10, 10) for the target MC player, because human players often have various preferences in such RPGs. Also 5 battle settings were prepared and employed, as shown in Table II and Table V, because various situations are given in such RPGs. Totally, 20 combinations were tested.

TABLE V. SETTING-1, 3, 4, 5. PARAMETERS AND AVAILABLE SKILLS OF CHARACTERS

Setting 1 (many enemies)						
Character	group	HP	MP	ATK	DEF	Available skills
Hero-1		134	30	60	28	single attack, lesser heal, defense
Hero-2		102	80	44	32	single attack, group attack, lesser heal, greater heal, group heal, defense
Enemy-1, 2, 3	1	52	0	38	26	single attack
Enemy-4, 5, 6	2	52	0	38	26	single attack
Enemy-7, 8	3	52	0	38	26	single attack
Enemy-9, 10	4	52	0	38	26	single attack
Setting 3 (easier situation)						
Character	group	HP	MP	ATK	DEF	Available skills
Hero-1		138	30	62	30	single attack, lesser heal, defense
Hero-2		110	62	46	34	(the same as setting 1)
Enemy-1, 2	1	52	0	40	26	single attack
Enemy-3	2	56	0	56	56	single attack
Setting 4 (harder situation)						
Hero-1		142	32	66	36	single attack, lesser heal, defense
Hero-2		112	64	48	38	(the same as setting 1)
Enemy-1	1	84	0	84	20	single attack
Enemy-2	2	84	0	44	60	single attack
Enemy-3	3	60	0	48	26	single attack
Setting 5 (boss battle)						
Hero-1		160	36	74	48	single attack, lesser heal, defense
Hero-2		122	72	52	44	(the same as setting 1)
Enemy-1	1	120	0	54	26	single attack
Enemy-2	2	222	0	80	40	single attack
Enemy-3	3	102	32	52	24	single attack, lesser heal

In all settings, only hero-2 selects the action according to the given preference vector, and the other characters play by a fixed rule. The proposed method records the states and actions of hero-2, and estimates his preference vector. Battles are done 8 times in a row (but HPs and MPs are initialized in each game), the proposed method estimates the preference after a half, 1, 3, 5 and 8 games, and the progress of estimation (accuracy improvement) is checked. Since results can be fairly affected by the randomness of the game itself and the algorithm, 20 trials (160 battles) are done using different random seeds, for each combination of target vector and battle setting.

First, Figure 6 shows the estimated weights after 1 game (left) and that after 8 games (right), when the target weight vector is $(1, 10, 10)$ and the battle setting-2 is used. In the case of only 1 game, the estimated weight vectors are widely distributed. On the other, hand after 8 games, the estimated weight vectors are almost all near the target vector $(1, 10, 10)$, and they seem to be on an edge, such as the one in Figure 5.

Evaluations should be done not only by the estimated vectors, but also by the matching rate defined in VI-C. In the above case, the averaged matching rate of the target MC player (to himself) is 84.2%. The averaged rate of the MC player with the estimated weight vectors is 69.9% after a half game, 77.5% after 1 game, and 83.5% after 8 games, respectively. It shows that the estimation accuracy was improved gradually, finally reaching almost the same level as the target player itself.

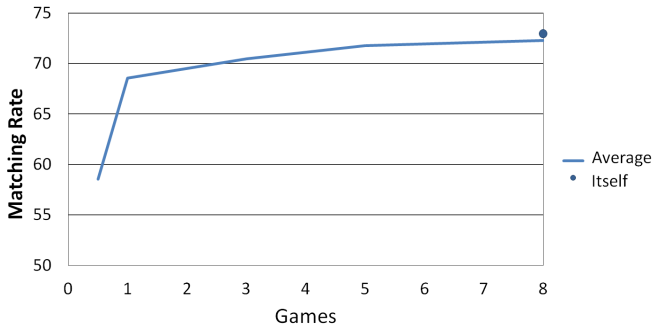

Fig. 7. Progress of averaged matching rate, total.

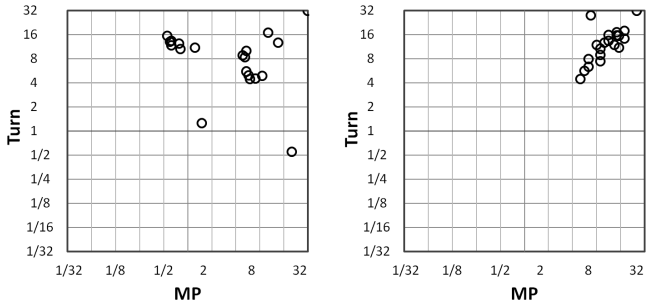

Fig. 6. Estimated weights after 1 game (left) and 8 games (right), when the target vector is $(1, 10, 10)$.

Figure 7 shows the averaged values of 20 combinations of battle settings and weight vectors (total of 400 trials). The averaged matching rate of the target MC player to himself is 72.4%, that of the proposed method is under 60% after a half game but over 70% after 8 games. Also Figure 8 shows the averaged values, separated between each battle setting. Though in some cases matching rates are relatively lower and in some cases higher, the gap between the matching rate of the target player itself and that of the proposal method after 8 games is only about 3% at max. It can be concluded that the performance of the proposed method is robust to the battle settings.

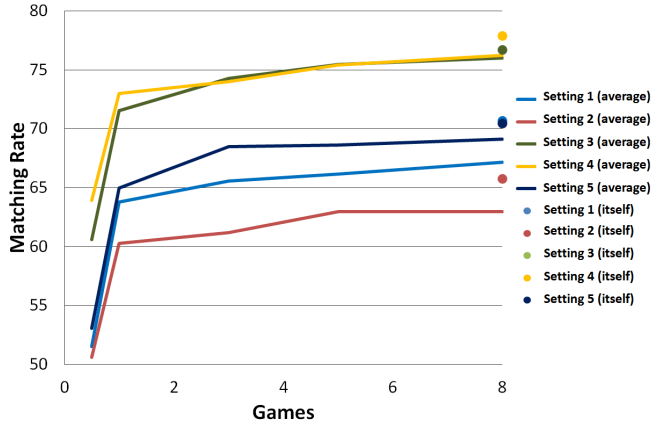

Fig. 8. Progress of averaged matching rate, for each battle setting. With different 5 battle settings, 4 target weight vectors and 160 battles (3200 battles totally).

IX. Evaluation Experiment Using Human Subjects

The experiment in the previous section was done in an ideal case where the target players had the same forms of preference funsctions as we had assumed.

Thus, we did the experiments with subjects in order to evaluate the performance of our method for human players.

A. Design

Firstly, each subject plays two battles only for becoming familiar to the rules and settings of the game. Secondly, the subject plays four sets of battles and each set consists of eight battles. At the start of each set, a direction is ordered to the subject:

1) Please win while keeping as much HPs as possible
2) Please win while preserving as much MPs as possible
3) Please win quickly
4) Please win quickly, and while preserving MPs

The former four battles of each set are the "learning phase". Here the subject controls both hero-1 and hero-2, the actions are recorded, and the preference vector is estimated by the proposed method.

The latter four battles of each set are the "evaluation phase", where the subject controls hero-1, and an MC player controls hero-2 without incremental learning. The MC player uses the estimated preference vector only in two games, and uses two fixed vectors $(1, 0.3, 3)$ and $(1, 4, 0.25)$ in the other two games, for comparison. At the end of each battle, the subject evaluates the degree of his/her satisfaction with the team-mate computer player, on a evaluation scale of five grades.

B. Results

Setting-2 (Table II) was employed through all the battles. The calculation of the feature x_{HP} was slightly modified in this experiment. It is measured not from the *final* HPs from the *averaged* HPs among the simulations.

10 human subjects attended this experiment. All of them are experienced at playing command-based RPGs. They used a Windows GUI, and about 1 to 2 hours were needed for the total of 34 battles.

The results are summarized in Table VI. For all the four kinds of given directions, the evaluation scores for the estimated preferences were better than that for the two fixed preferences (Speedy/MP-preserving). In the case of HP-keeping direction, since the fixed preferences were unsuitable for the direction, then the result is reasonable. In the case of MP-preserving direction, the fixed preference $(1, 4, 0.25)$ was better than $(1, 0.3, 3)$, but the estimated preference was much better. Probably, the fixed preference $(1, 4, 0.25)$ was not enough, and more extreme preference such as $(1, 20, 0.05)$ was preferred.

The last direction was intentionally designed to be mixed and vague. Human subjects understood the direction in different ways, then the estimated vectors were widely distributed for example $(1, 11, 24)$ and $(1, 23, 23)$. We believe that the proposed method is useful because such individual differences are very frequent in actual RPGs.

TABLE VI. SATISFACTION FOR TEAMMATE COMPUTER PLAYERS

Direction	MC player	Average scores
Keeping HPs	Proposed method	3.8
	MP-preserving: $(1, 4, 0.25)$	2.9
	Speedy: $(1, 0.3, 3)$	3.2
Preserving MPs	Proposed method	3.4
	MP-preserving: $(1, 4, 0.25)$	3.0
	Speedy: $(1, 0.3, 3)$	2.1
Speedy	Proposed method	4.2
	MP preserving: $(1, 4, 0.25)$	2.5
	Speedy: $(1, 0.3, 3)$	4.0
Speedy and preserving MPs	Proposed method	4.0
	MP-preserving: $(1, 4, 0.25)$	3.0
	Speedy: $(1, 0.3, 3)$	2.7

X. CONCLUSION

In some games such as RPGs, not only "winning" but also many sub-goals are implicitly sought by human players. In this paper, we proposed a method to estimate the preferences about such sub-goals from the player's actions, and to cooperate well as a team-mate by respecting these preferences. The preferences were modeled by a parameterized function, and the parameters were optimized to each player, by using a multi-strategy Monte-Carlo method. The effectiveness of the proposed method was confirmed through several series of experiments, using artificial players and using human subjects. We showed that the proposed method can estimate almost the exact preferences of the artificial players after only 8 games, and that the proposed method can play with human players without generating dissatisfaction.

ACKNOWLEDGEMENT

This research was supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (C), (No. 26330417). The authors also would like to thank participants in our experiments.

REFERENCES

[1] S. Bakkes, P. Spronck and E. Postma "TEAM : The Team-Oriented Evolutionary Adaptability Mechanism," in *Proc. Int. Conf. on Entertainment Computing*, 2004, pp.273-282.

[2] N. Fujii, Y. Sato, H. Wakama, K. Kazai and H. Katayose, "Evaluating Human-like Behaviors of Video-Game Agents Autonomously Acquired with Biological Constraints," in *Int. Conf. on Advances in Computer Entertainment Technology*, 2013, pp. 61-76.

[3] "Mario AI Championship 2012," [Online]. Available: http://www.marioai.org/.

[4] M. Bernacchia and J. Hoshino, "AI platform for supporting believable combat in role-playing games," in *Proc. 19th Game Programming Workshop in Japan*, 2014, pp. 139-144.

[5] M. Bernacchia and J. Hoshino, "Believable fighting characters in role-playing games using the BDI model," in *31st meeting Game Informatics Research Group*, 2015, pp. 1-8.

[6] K. Ikeda and V. Simon, "Efficiency of static knowledge bias in monte-carlo tree search," in *Computers and Games*, 2014, pp. 26-38.

[7] J. van der Wal, "Stochastic dynamic programming," Ph.D. dissertation, Methematisch Centrum, Amsterdam, Nederland, 1980.

[8] P. Jansen, "Using knowledge about the opponent in game-tree search," Ph.D. dissertation, Carnegie-Mellon University, Pennsylvania, U.S., 1992.

[9] D. Carmel and M. Shaul, "Learning models of opponent's strategy in game playing," *in Proc. 1993 AAAI Fall Symp. Games: Planning and Learning*, pp. 140-147.

[10] H. Iida, J. W. H. M. Uiterwijk and H. J. van den Herik, "Opponent-Model search," Maastricht Univ., Techn. Rep. CS-93-03, 1993.

[11] K. Hoki, "Optimal control of minimax search results to learn positional evaluation," *in Proc. 11th Game Programming Workshop*, 2006, pp. 78-83.

[12] A. Y. Ng, S. Russel, "Algorithms for Inverse Reinforcement Learning ," *in Proc. 17th Int. Conf. Machine Learning*, 2000, pp. 663-670.

[13] J. V. Neumann and O. Morgenstern, "Theory of games and economic behavior," Princeton Unive. Press, 1944.

[14] Y. Tsuruoka, D. Yokoyama and T. Chikayama. (2002, Sept.). Game-tree Search Algorithm based on Realization Probability. *International Computer Games Association Journal. 25(3)*, pp. 145-152.

[15] S. Namai and T. Ito, "A trial AI system with its suggestion suggestion suggestion of Kifuu (playing style) in Shogi," in *2010 Int. Conf. Technologies and Applications of Artificial Intelligence*, 2010, pp.433-439.

[16] R. Coulom. (2007.). Computing Elo ratings of move patterns in the game of Go. *International Computer Games Association Journal. 30(4)*, pp.198-208.