IEICE TRANS. INF. & SYST., VOL.E97–D, NO.4 APRIL 2014

PAPER

An Intelligent Fighting Videogame Opponent Adapting to Behavior Patterns of the User****

Koichi MORIYAMA[†a)], ***Member***, **Simón Enrique ORTIZ BRANCO**[††*], **Mitsuhiro MATSUMOTO**[††**], **Ken-ichi FUKUI**[†], ***Nonmembers***, **Satoshi KURIHARA**[†***], ***and*** **Masayuki NUMAO**[†], ***Members***

SUMMARY In standard fighting videogames, users usually prefer playing against other users rather than against machines because opponents controlled by machines are in a rut and users can memorize their behaviors after repetitive plays. On the other hand, human players adapt to each other's behaviors, which makes fighting videogames interesting. Thus, in this paper, we propose an artificial agent for a fighting videogame that can adapt to its users, allowing users to enjoy the game even when playing alone. In particular, this work focuses on combination attacks, or combos, that give great damage to the opponent. The agent treats combos independently, *i.e.*, it is composed of a subagent for predicting combos the user executes, that for choosing combos the agent executes, and that for controlling the whole agent. Human users evaluated the agent compared to static opponents, and the agent received minimal negative ratings.

key words: *entertainment computing, adapting agent, pattern matching, reinforcement learning*

1. Introduction

In recent decades, videogames have been widely played in the world. Videogames consist of many genres such as role-playing games, action games, and simulation games. Nowadays, they are spreading outside of entertainment such as serious games and "edutainment". Videogames are good applications of computer science and information technology because they are software produced by synthesizing many techniques of them.

One of the popular genres of games with new titles being released every year is the fighting game. It is basically a simulation of hand-to-hand combat, in which fights are carried out in a manner similar to boxing matches: usually there are two participants, there are several rounds with a given time limit, etc. The participant who has lowered the energy of the opponent to zero by means of attacks is the winner of the round.

All fighting videogames are designed to be played by at least two users competitively, *i.e.*, users playing against one another by controlling each character. However, these games also have the possibility of being played by only one user. In this case, the machine will take control of the opponent character. Users generally choose to play against the machine in order to advance the game story, to practice, or because there is nobody around to play with. But, if given the option, users usually prefer to play against other users.

One of the reasons users prefer to play against other users could be that the opponent controlled by the machine in solo mode is usually uninteresting. This is not to say that it is easy to defeat, since the machine can execute complicated attacks and respond quickly. Rather, the adaptability of human players makes battles interesting compared to the machine. Meanwhile, the machine opponent used so far is just sophisticated enough to have the user feel that the opponent is reasonably smart [3].

When two users play against each other, they usually fight a battle beyond quick button mashing. Each user has a strategy he/she follows and patterns he/she developed. The fun part of the game is trying to learn each other's technique and predict the future actions of his/her opponent. Hence, we are aiming to produce an artificial opponent that adapts to its users *by simulating the learning and prediction*. Such an adapting opponent will let all users at different levels of expertise enjoy the game even when playing alone.

In this work, we propose an agent controlling such an adapting opponent. In particular, this work focuses on combination attacks, or *combos*, that need techniques to execute infallibly but deal greater damage than simple attacks like punches and kicks. In addition, to defend from combos, the player has to take actions prescribed by each combo. Thus, the agent has to learn what combos the user can execute and predict what combos the user will execute next. Also, it has to adjust its own executable combos to have them balance with the user's executable ones. Hence, the agent consists of three subagents: a subagent for learning and predicting combos the user executes, that for choosing combos the agent executes, and that for controlling the whole agent. In the experiment, human users at different levels of expertise evaluated the degree of fun the agent added to playing the game.

This paper consists of seven sections. Section 2 introduces standard fighting videogames and their components. We present the model of our adapting agent for a fighting videogame in Sect. 3. Section 4 shows the result of experi-

Manuscript received July 23, 2013.
Manuscript revised November 27, 2013.

[†]The authors are with the Institute of Scientific and Industrial Research, Osaka University, Ibaraki-shi, 547–0047 Japan.
[††]The authors are with the Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan.
[*]Presently, with PlatinumGames, Inc.
[**]Presently, with Mitsubishi Electric Corporation.
[***]Presently, with Graduate School of Information Systems, The University of Electro-Communications.

[****]This article is an extended version of our workshop and conference papers [1], [2].

a) E-mail: koichi@ai.sanken.osaka-u.ac.jp

DOI: 10.1587/transinf.E97.D.842

ment in which human users evaluated the agent compared to static opponents. We discuss the result in Sect. 5. After we briefly see several related works in Sect. 6, Sect. 7 concludes this paper.

2. Fighting Videogame

Typical fighting videogames are one-on-one games. In solo mode the user is in control of one character, and the machine controls the opponent. Each character starts with a predefined amount of health points (HP); when their HP reaches zero they lose. The objective is to defeat the other character first. Usually the fight is decided after several rounds each of which has a predetermined time limit. Fighting videogames have several stages where the fights occur. The characters can walk back and forward, jump and crouch in a 2D plane of the stage. If the stage has edges, falling from the edge typically means immediately losing the round.

Although different fighting videogames vary in the details, the set of available actions can be described as $\mathsf{A} \cup \mathsf{D} \cup \mathsf{M} \cup \mathsf{C} \cup \mathsf{B}$. A is the set of *simple attacks*, *e.g.*, punch, kick, and projectile-like long range special attacks (from now on we will abbreviate these actions as `p`, `k` and `s` respectively). Simple attacks deal moderate damage. D is the set of defensive actions, or *blocks*. Blocks guard the character from simple attacks. M is the set of movements the players use to navigate the character. C and B are the sets of *combos* and *combo-breakers*, respectively. They are key actions in this work and explained in detail in the next subsections.

2.1 Combos

Combos, short for combination attacks, are a common game design element found in most modern fighting videogames. A combo deals greater damage than simple attacks. Different games have a different approach to combos. We deal with combos of the form of predefined sequences of simple attacks successfully executed within a brief time limit.

For instance, let us consider the combos shown in Table 1. The sequence of four punches in a row `pppp` is a combo. When a character successfully connects four punches the opponent will receive the extra amount of damage defined for this combo in addition to the normal amount of damage of each simple attack. After the first two actions of a combo are successfully connected, the rest of the actions of this combo *cannot be blocked by the block actions*.

Table 1 Example of combos.

ID	Actions	Damage	ID	Actions	Damage
1	`pppp`	15	7	`kppk`	20
2	`pppk`	20	8	`kpps`	25
3	`ppps`	25	9	`kspp`	15
4	`pkpp`	20	10	`ksps`	20
5	`pkpk`	15	11	`kpsp`	30
6	`pkps`	25	12	`kkkk`	30

2.2 Combo-Breakers

Combo-breakers are a way of blocking or counter-attacking a combo. For some games, including the case we deal with, *combo-breakers are the only ways to defend from a combo*. The combo-breaker is one predefined action or sequence of actions that can be executed by the character receiving a combo. Each combo $c \in \mathsf{C}$ has at least one combo-breaker $b \in \mathsf{B}$. In order to break a combo (*i.e.*, to counter-attack it), the corresponding combo-breaker should be executed before the combo is completed. When a combo is broken, the character receiving the combo will not receive the extra damage, instead the character executing the combo will receive it.

An example will clarify this concept. Let us use again the combos in Table 1. The sequence `pppp` is defined as a combo. Let us assume that this combo has the action `p` as its combo-breaker. Suppose the attacker is executing this combo. If the defender executes `p` before the attacker's fourth `p`, then the combo will be broken and the attacker will receive the extra damage 15. Instead, if the defender cannot execute `p` before the attacker's fourth `p`, or if the defender's last action is not `p`, the combo will not be broken and the defender will receive the extra damage of the combo.

3. Three-Subagent Adapting Architecture

Our goal is to produce an agent for fighting videogames that can adapt to its users' behaviors, or fighting styles, allowing users to enjoy the game even when playing alone. The agent must adapt automatically to the level of the user, *i.e.*, it must become a difficult opponent if the user is an expert, and it must become an easy opponent if the user is a beginner.

Playing a fighting videogame can be thought of as dealing with three tasks: executing simple attacks plus moving, executing combos, and executing combo-breakers when receiving a combo. According to the three tasks, we divide the agent into three subagents: Main Subagent (MSA), Executing-Combo Subagent (ECSA) and Receiving-Combo Subagent (RCSA) (Fig. 1). MSA will move the character and execute simple attacks. When deemed appropriate, it will pass the control to one of the other subagents. ECSA

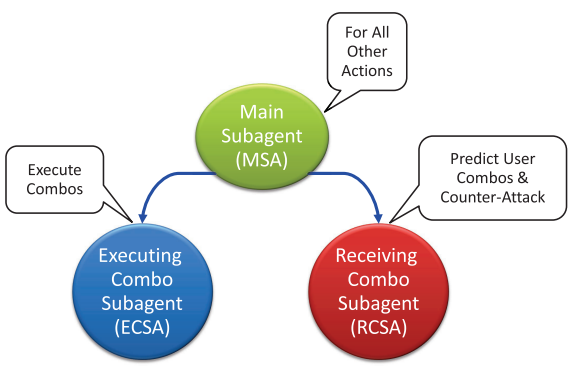

Fig. 1 Three subagent architecture.

will execute combos having similar difficulty to those executed by the user. RCSA will learn the fighting style of the user and try to execute the appropriate combo-breaker.

The agent composed of the mentioned subagents will be able to move away from the user or get close to him/her depending on whether the user will attack or not, execute combos appropriate to the level of the user, and learn the user's combo patterns to respond adequately. Note that dividing the tasks among subagents should help the agent learn each task in less time.

3.1 Main Subagent (MSA)

MSA is in charge of executing simple attacks, blocks, and moving. When deemed appropriate, MSA passes the control to one of the other subagents. MSA is modeled as a Profit-Sharing agent [4], shown in Fig. 2.

A *round* is a period from the start of a battle until the winner is determined or the predetermined time passes. From a viewpoint of each player, a round is divided into *turns* each of which is a period from one action to the next one of the player. At the t-th turn of the round n, MSA first recognizes the environment as a state s_t and looks up recorded weights $w_{n-1}(s_t, a^i)$ of all actions a^i available in s_t. After that, MSA chooses an action a_t with the probability calculated from the weights using Boltzmann equation [5], with *temperature* τ:

$$P_{s_t}(a^i) = \frac{\exp(w_{n-1}(s_t, a^i)/\tau)}{\sum_k \exp(w_{n-1}(s_t, a^k)/\tau)}. \quad (1)$$

The agent executes a_t and records the pair (s_t, a_t). The available actions are those defined in the videogame in question and passing control of the character to ECSA/RCSA. After the action has been executed, or ECSA/RCSA has executed its action, MSA resumes control of the character.

At the end of the round, MSA receives a reward R_n from the environment and updates the weight of all recorded pairs (s_t, a_t) of this round by the following rule. T is the last turn in the round.

$$w_n(s_t, a_t) \leftarrow w_{n-1}(s_t, a_t) + R_n \cdot \gamma^{T-t} \quad (1 \le t \le T). \quad (2)$$

MSA receives higher positive rewards when the difference of the final HPs of the agent and the user is small, although negative rewards are given when the difference is significant. This reinforces actions that lead the agent to behave in such a way that it is not too difficult nor too easy for the user. That is, we are reinforcing actions that put the agent at the same level of the user.

```
MSA():
  while true do
    s := the current state of the game;
    a := an available action in s chosen according to
      the weight w(s,a); // See Eq.(1)
    execute the action a;
    record the pair (s,a);
    if (the round is finished)
      update w(s,a) of the recorded pairs (s,a)
        by the obtained reward; // See Eq.(2)
      exit;
    endif
  done
```

Fig. 2 Pseudo-code of MSA.

3.2 Executing-Combo Subagent (ECSA)

ECSA has the responsibility of choosing combos and executing them. In order for the agent as a whole to be at the same level of the user, the combos the agent executes must also be on a level close to that of the user.

Since the agent must act in real-time during a round, ECSA randomly selects a combo from its combo set $C_A \subseteq \mathsf{C}$ and executes the selected one when invoked. Therefore, the problem is how to create C_A. If we consider the set of combos used by the user, $C_U \subseteq \mathsf{C}$, the goal of ECSA is to create C_A of similar difficulty.

To create C_A, we need metrics to order sets by their difficulty. We use the following three metrics: *ratio of used combos*, *indistinguishability of combos*, and *entropy of combo-breakers*. In the following definitions of metrics, $C_X \subseteq \mathsf{C}$ is a set of combos.

Ratio of used combos: A better user would execute a wider variety of combos because it would make it difficult for the opponent to predict the combo-breakers. Hence, the ratio of used combos is a valid metric:

$$\mathit{used\text{-}ratio}(C_X) = \frac{|C_X|}{|\mathsf{C}|}. \quad (3)$$

Indistinguishability of combos: Since the combo-breaker must be executed *before* the last action of the combo, a set of combos that are indistinguishable by the initial actions is more difficult than a set where the combos can be distinguished by their initial actions. Let $C_X^s \subseteq C_X$ be a set of solitary combos that do not share the initial actions with others in C_X. Then, this metric can be formalized as follows:

$$\mathit{inds}(C_X) = \frac{|C_X| - |C_X^s|}{|C_X|}. \quad (4)$$

Entropy of combo-breakers: The set of combos sharing a combo-breaker is easier than that of combos having different combo-breakers, because a player playing against the former need not decide which combo-breaker should be executed. Hence, the entropy of the set of *distinct* combo-breakers, B_X, of C_X is a valid metric:

$$\mathit{entr}(C_X) = \frac{-\sum_{b \in B_X} P(b) \log P(b)}{\log |B_X|} \quad (5)$$

where $P(b)$ is the probability of randomly choosing a combo-breaker b out of the combo-breakers of C_X.

ECSA first creates a combo set containing m combos, whose combo-breakers and initial actions are different (high combo-breaker entropy, low indistinguishability). We consider that such an initial set is not too difficult nor too easy.

```
adaptC_A():
  if (used-ratio(C_A) > used-ratio(C_U) + Δ)  delete();
  elif (used-ratio(C_A) < used-ratio(C_U) - Δ)  add();
  else swap();

delete():
  for (i:=0; i < max_iter; i++)
    a := a combo in C_A chosen randomly;
    if (|entr(C_A\{a})-entr(C_U)| < |entr(C_A)-entr(C_U)|
      or |inds(C_A\{a})-inds(C_U)| < |inds(C_A)-inds(C_U)|)
      C_A := C_A\{a}; return;
    endif
  endfor

add():
  for (i:=0; i < max_iter; i++)
    c := a combo chosen randomly from all available ones;
    if (|entr(C_A∪{c})-entr(C_U)| < |entr(C_A)-entr(C_U)|
      or |inds(C_A∪{c})-inds(C_U)| < |inds(C_A)-inds(C_U)|)
      C_A := C_A∪{c}; return;
    endif
  endfor

swap():
  for (i:=0; i < max_iter; i++)
    a := a combo in C_A chosen randomly;
    c := a combo chosen randomly from all available ones;
    if (|entr((C_A∪{c})\{a})-entr(C_U)| < |entr(C_A)-entr(C_U)|
      or |inds((C_A∪{c})\{a})-inds(C_U)| < |inds(C_A)-inds(C_U)|)
      C_A := (C_A∪{c})\{a}; return;
    endif
  endfor
```

Fig. 3 Pseudo-code for adapting C_A.

After finishing a round, this combo set is partially adapted to that of the user by the algorithm presented in Fig. 3. Δ defines the level of tolerance in the difference of sizes of the sets, and `max_iter` limits the number of tries. Note that ECSA does not use the damage dealt by each combo as the metrics because it restricts the choice of combos in the adaptation.

3.3 Receiving-Combo Subagent (RCSA)

RCSA chooses the proper combo-breaker for the combos executed by the user. Since the combo-breaker must be executed before the last action of a combo, the problem becomes predicting the full combo from its beginning. How do we do? It is reasonable to expect that users develop routines of maneuver in fighting videogames because they allow the user to execute a series of attacks quickly without having to decide each attack, and such quick attacks are successful in giving damage to the opponent. Therefore, for the prediction problem, it is useful to extract patterns of the combos previously executed by the user.

For example, suppose the user can execute only four combos, *i.e.*, $|C_U| = 4$ (ID $= 1, 2, 3, 4$), and he/she took combos in the following order: 1, 2, 3, 4, 1, 3, 4, 2, 3, 4, 1, 2, 3, 4, 2, 1, 4, 3. Then, if we can find any patterns in the sequence, they are useful to predict the current combo the user is executing. From the above example, we get the following subsequences (when minimum support $= 2$): $\langle 3\rangle_5$, $\langle 4\rangle_5$, $\langle 1\rangle_4$, $\langle 2\rangle_4$, $\langle 3, 4\rangle_4$, $\langle 2, 3, 4\rangle_3$, $\langle 1, 2, 3, 4\rangle_2$, $\langle 2, 3, 4, 1\rangle_2$, $\langle 3, 4, 1\rangle_2$, $\langle 3, 4, 2\rangle_2$, $\langle 4, 1\rangle_2$, $\langle 4, 2\rangle_2$ (subscripts show frequencies). In this case, if the user executed Combo 3 previously,

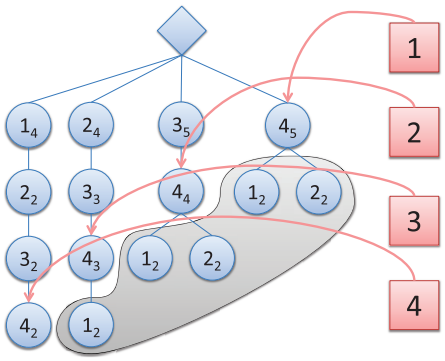

Fig. 4 An example of combo tree and pointers tracking the tree (subscripts in nodes show frequencies). Gray area shows the candidate combos in the example (See text).

we can infer that he/she will execute Combo 4 with a high probability and it is worth to execute a combo-breaker for Combo 4 if the beginning of the executing combo meets it. Although any sequential pattern extraction techniques can be used, we utilized Substring Tree [6].

After extracting frequent subsequences, we transform them into a *combo tree* where each node shows a combo with its frequency. We start with the empty tree, and insert each frequent subsequence of combos beneath the empty root node. Subsequences with the same beginning will share parts of the tree. If the extracted subsequences are those in the above example, the tree will be the one shown in Fig. 4.

Since the user can have maneuver routines of various lengths, the agent tracks all possible patterns up to length l simultaneously. To do so, the agent traverses the tree following the combos recently executed by the user during the round. The agent uses l pointers to the tree at the same time. Each of the l pointers tracks a sequence of size $1, 2, \ldots, l$ combos. In the above example, if the user executed Combos 1, 2, 3, and 4 in this order, then the first pointer will be at the rightmost branch of Fig. 4, the branch that starts with 4; the second pointer will point at Node 4 in the middle-right branch, below 3; the third pointer would point to Node 4 in the middle-left branch, below 3 below 2; etc. If the sequence of combos executed by the user does not match any sequences of combos of length k in the tree, then the k-th pointer will be null.

The combo-breaker is decided following the algorithm of Fig. 5. What the agent basically does is, for all the pointers pointing to a valid node, the agent searches combos beneath them that match the beginning with the combo being executed now. It chooses a combo-breaker stochastically by their relative frequency among the combos found under the pointers. Suppose a situation where the combos are defined as in Table 1 and the user executes actions ppp. Then, the agent has to predict the combo the user is executing now from the combos that meet the leading actions, *i.e.*, Combos 1, 2, and 3. If the combo tree and the pointers extracted from the user's previous behavior are those in Fig. 4, the agent collects the frequencies from the candidate combos that are beneath the pointers, *i.e.*, combos in the gray area of Fig. 4:

```
decide_combo_breaker():
  combo_head := first actions of the combo being
    executed;
  combo_seq[l] := last l combos executed by the user,
    from recent to old;
  candidate := {};
  for ( i: 1 to l )
    p[i] := track(combo_seq,i);
  endfor
  for all p[i] ≠ null
    for all children c of p[i]
      if (c's first actions = combo_head)
        add c to candidate;
      endif
    endfor
  endfor
  return a combo-breaker for the combo randomly
    chosen from candidate;

track(combo_seq, i): // See Fig.4
  node := root of the combo tree
  for ( j: i to 1 )
    if (combo_seq[j] ∈ children of node)
      node := a child matching combo_seq[j]
    else
      node := null
      break;
    endif
  endfor
  return node
```

Fig. 5 Pseudo-code for deciding a combo-breaker.

two times of Combo 1 from Pointer 3, two times of Combo 1 and two times of Combo 2 from Pointer 2, and two times of Combo 1 and two times of Combo 2 from Pointer 1. After that, the agent calculates the likelihood of Combo 1 by $(2+2+2)/(2+2+2+2+2) = 0.6$ and that of Combo 2 by $(2+2)/(2+2+2+2+2) = 0.4$. Note that it does not calculate the likelihood of Combo 3 because it is not in the candidates. Then, the agent takes the combo-breaker for Combo 1 with probability 0.6 and that for Combo 2 with probability 0.4.

Note that RCSA also does not use the damage of combos because it does not affect the prediction itself. Even though, for example, the user prefers combos dealing large damage, it will appear in the combo tree.

4. Experiment

We developed a simple fighting videogame using Crystal Space 3D[†] to test the proposed adapting agent. Figure 6 shows a screenshot of the videogame.

The fighting videogame has the following characteristics: the fights occur in a 2D plane; the characters have a height of 3.5 units and a width of 2 units; the stage is a finite platform with a length of 42 units, falling from the platform equals losing the round; there is no time limit; there is one round per fight (hence we use the term "round" in both meanings hereinafter); the initial HP of the characters is 200; the set of simple attacks (A) contains punch (p), kick (k), and special attack (s), the last one being a long range projectile attack that travels 15 units per second; the characters cannot do anything for 0.3 seconds after a punch or a special attack, while 0.4 seconds after a kick; each of the simple attacks deal one point of damage; the set of defenses (D) contains one action: block; while blocking, simple attacks do not have effect; the set of combos (C) is listed in Table 1; for a combo to be valid, each action must be executed within 0.5 seconds of the previous one; the combo-breaker of a combo is defined as its *last action*; if the combo-breaker is valid, the character executing the combo receives its damage; and the set of movements (M) contains move to the right, move to the left, jump and crouch.

[†]http://www.crystalspace3d.org/

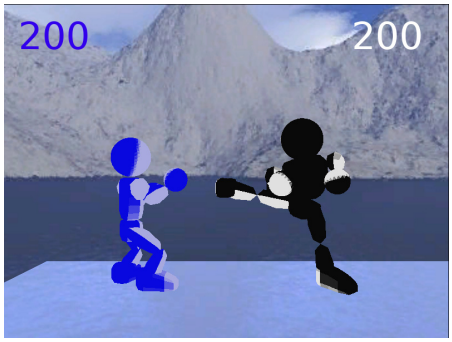

Fig. 6 Fighting videogame.

We first show how well RCSA works in the fighting videogame. Next, we show how well the (whole) proposed agent (Fig. 1) makes the videogame enjoyable.

4.1 Experiment 1: How Well RCSA Works

In this experiment, we verified how well an agent using RCSA can break the user's combos by predicting them properly. We first constructed an agent that could recognize the combos and execute combo-breakers chosen by a choice method such as RCSA. In order to evaluate the method itself, the agent *always* executed a combo-breaker when receiving a combo.

We used two other combo-breaker choice methods for comparison: *random* and *accumulation*. *Random* randomly chose a combo-breaker. *Accumulation* memorized the combos the user had executed in the past and chose a combo-breaker stochastically based on the frequency of the user's combos that fitted the initial actions.

As users who played against the agents having one of the combo-breaker choice methods, we constructed two "simulated" ones. The first one, named "single", executed combos obeying the pattern 6–4–9–11–3–5, which means that the simulated user first executed Combo 6, then Combo 4, etc., and after Combo 5, it executed Combo 6 again. The other simulated user, named "alternate", first followed the above pattern for 10 rounds and, after that, it followed another combo pattern 1–3–10–2–8–3 for 10 rounds. Then, it alternated these patterns every 10 rounds. The combos in the patterns were chosen randomly and, needless to say, the agents did not know the patterns in advance.

Figure 7 shows the accuracy of the combo-breaker choice methods. The accuracy is defined as the number of successfully executed combo-breakers divided by the num-

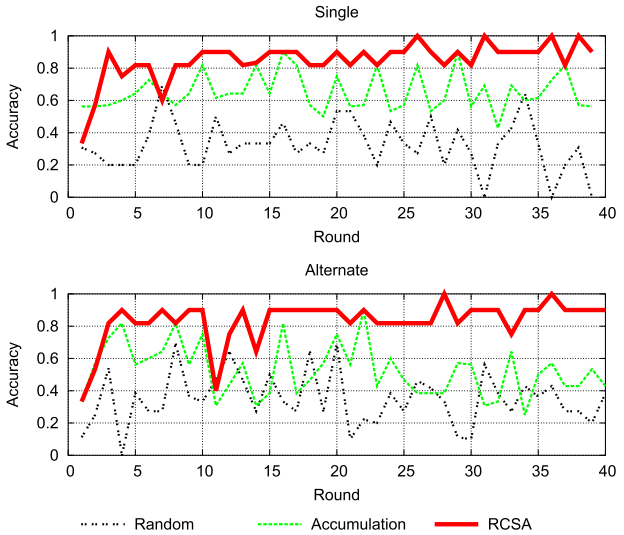

Fig. 7 The accuracy of combo-breaker choice methods, which shows how well they could break combos executed by simulated users: the "single" (above) and the "alternate" (below).

ber of combos presented to the agent. In RCSA, the minimum support of frequent sequences was set to two, and five pointers were used in the combo tree. The ranking of the combo-breaker choice methods was as follows: RCSA, *accumulation*, and *random*. In particular, when playing against "alternate", RCSA obtained high accuracy, whereas *accumulation* degraded because it was confused by the alternation of patterns. This result shows that RCSA works well when the user has patterns in combo execution.

4.2 Experiment 2: How Well the Agent Makes the Game Enjoyable

We conducted an experiment in which human users played against the (whole) proposed agent (Fig. 1) to show how well the agent makes the videogame enjoyable for each user. The proposed agent had the following parameters:

States: To keep the design of the agent simple, we discretized the world state as follows. These were selected because they provided enough information to the agent to make intelligent decisions: (a) crouching or not (agent/user), (b) jumping or not (agent/user), (c) receiving a combo or not (agent), (d) executing a simple attack or not (user), (e) blocking or not (agent/user), (f) at an edge of the platform or not (agent), (g) HP $<$ 30 or not (user), (h) the distance between the user and the agent, discretized in eight sections: ≤ 0.25, ≤ 0.50, ≤ 2.00, ≤ 2.60, ≤ 4.00, ≤ 10.00, ≤ 24.50 and > 24.50, (i) the distance from the agent to the closest special attack thrown by the user, discretized in three sections: ≤ 0.50, ≤ 2.60 and > 2.60, and (j) the difference in HP between the user and the agent, rounded to tens.

Actions: The available actions were those available in the game, plus ECSA, RCSA, and stay. Instead of the actions right and left, the agent used "approach" and "withdraw". Approach, withdraw, crouch, and block were executed for 0.1 seconds. Stay had a duration of 0.4 seconds. All the other actions lasted as long as it took to fully execute them.

Table 2 Rewards.

HP difference	Reward	HP difference	Reward
< 25	+1.00	< 125	−0.25
< 50	+0.75	< 150	−0.50
< 75	+0.25	< 175	−0.75
< 100	−0.10	≥ 175	−1.00

Rewards: In order to have the agent adapt to the level of the user, the reward was defined as Table 2. It was given to the agent when a round was finished, regardless of which had won.

Others: In MSA, γ was fixed at 0.99 [†] and τ was fixed at 1.0. In ECSA, $m = 3$, $\Delta = 0.1$, and `max_iter` $= 20$. The setting of RCSA was identical to that of Experiment 1.

We used two versions of our adapting agent: `adapt0` and `adaptT`. The `adapt0` agent was as explained in Sect. 3. The `adaptT` agent was structurally the same as `adapt0`, but it had been trained by a certain user (who was out of the experiment) for 20 rounds in advance.

For a comparison purpose we developed three static agents: `weak`, `strong` and `trained`. The `weak` agent was very easy to defeat; 50% of its action were to stay; it only executed Combos 1 and 11, which meant the combo-breaker was always `p`. On the other hand, the `strong` agent was very difficult to defeat; it always got close to the user, executed one of all available combos randomly whenever close enough, and stochastically chose combo-breakers by *accumulation* used in Experiment 1. However, it discarded the memory after every round. The `trained` agent was identical to the `adaptT` in the beginning, but did not adapt at all during the experiment.

We asked 30 real users to play the game. They were of different nationalities, genders, ages, and at different levels of expertise at playing videogames. Before the experiment, they had a training session until they got used to the game. In the training session, they played against a static agent specially designed for the session, which executed simple attacks and Combos 1, 10 and 12 randomly.

In the experiment, the users would fight against the above five types of agents presented in a random order. They did not know the characteristics of the agents. They were asked to play 30 rounds against each agent, but after 15 rounds, they could quit whenever they were no longer having fun. After playing against an agent, they took a break for between 5 and 30 minutes (depending on the user) and started to play against the next agent. After playing against all of the agents, they filled in a questionnaire that asked to rate the overall fun and difficulty of each agent in 5-point

[†]For example, when T in Formula 2 is 300, $\gamma^T \simeq 0.05$. Therefore, even if the number of turns became a few hundreds, the update in Formula 2 is not negligible. Remember that the initial HP is 200.

Table 3 Rating result of "fun": Showing the number of users who gave the score and the average scores for each agent.

Score	weak	strong	trained	adapt0	adaptT
5 (fun)	2	14	4	8	7
4	3	3	13	5	9
3	6	3	8	12	10
2	4	5	4	5	4
1 (bored)	15	5	1	0	0
Avg. Score	2.1	3.53	3.5	3.53	3.63
SD	1.32	1.61	1.01	1.07	1.00

Table 4 Rating result of "difficulty": Showing the number of users who gave the score and the average scores for each agent.

Score	weak	strong	trained	adapt0	adaptT
5 (difficult)	0	29	0	0	0
4	1	1	7	8	8
3	1	0	11	8	9
2	2	0	9	13	10
1 (easy)	26	0	3	1	3
Avg. Score	1.23	4.97	2.73	2.77	2.73
SD	0.68	0.18	0.94	0.90	0.98

scales and to describe comments in a free form. Also, they were asked to rank the agents from the best to the worst.

The results of the ratings are shown in Tables 3 and 4. From Table 3, we can see that many users had fun when playing against the strong agent, but, in the mean, the adaptT agent gave slightly more fun to the users. It is because, as shown in Table 4, all of the users answered that the strong was very difficult to defeat. Remember that the users had different levels of expertise at playing videogames. Expert users who were good at playing videogames thought that it was challenging to defeat the agent, but beginners thought that it was impossible to defeat, and then, they abandoned the game. Indeed, Table 3 shows that one-third of users gave 1 or 2 to the strong, that is, it was not interesting for them to play against it. The free-form comments of the users also support it, *i.e.*, many users complained about the difficulty of the strong: how it attacked too fast, how it executed a lot of combos, and how difficult it was to defeat it. On the other hand, no user gave 1 and only four gave 2 to the adaptT. That is, most users enjoyed playing against it.

We also compare the length of play against each agent in Table 5. Remember that the users were asked to play 30 rounds but could quit immediately after 15 rounds. Therefore, users who played 30 rounds may have wanted to play more. We see that the strong agent was played against in the longest time in the mean, but it is not so important because two-third of users stopped playing by 22 rounds regardless of the agents the users played against. On the other hand, it is notable that about one-third of users continued to play against the adaptT (and the strong) for 30 rounds but it was only three who did against the trained. This fact shows that it is important for the agent to adapt its level to the level of users, in order to let the users play long.

Moreover, Table 6 shows the ranking of the agents. The agent the most users ranked the best was the strong, followed by the adaptT. On the other hand, the agents the least

Table 5 Result of length: Showing how many users played against the agent until the round.

Round	weak	strong	trained	adapt0	adaptT
15	12	6	5	4	6
16	6	3	4	1	3
17	0	2	3	2	0
18	1	2	3	3	4
19	1	1	3	3	1
20	0	2	2	2	2
21	1	0	2	2	1
22	0	0	0	3	1
23	0	0	0	1	1
24	2	2	0	1	0
25	0	0	3	0	0
26	0	0	0	0	1
27	0	0	0	1	1
28	0	1	0	0	0
29	0	0	2	0	0
30	7	11	3	7	9
Avg. Rounds	19.73	22.43	20.2	21.77	21.93
SD	6.25	6.55	5.15	5.39	6.16

Table 6 Ranking result: Showing how many users put the agent into the rank order.

Rank	weak	strong	trained	adapt0	adaptT
1 (best)	0	13	4	6	7
2	4	5	8	5	8
3	4	1	8	8	9
4	2	8	7	9	4
5 (worst)	20	3	3	2	2

users ranked the worst were the adapt0 and the adaptT. In particular, if we see both the 4th and the 5th, the strong and the adapt0 were ranked by 11 users, although only six ranked the adaptT them.

Thus far, we saw the results from a collective view. Next, we investigate the results from each user's point of view; in particular, we compare the proposed agents (adapt0 and adaptT) and the trained because the results so far seem similar. As above mentioned, the trained has the same architecture of the proposed agents and trained for 20 rounds in advance, but it does not adapt to the user at all during the experiment. Then, the difference of them shows the effect of adaptation to the user.

Figure 8 shows the transition of final HP differences when a subject played against the three agents. The plots tell us that in most rounds the subject defeated the trained, whereas the subject sometimes won and sometimes lost when playing against the proposed agents. As a result, the average values of absolute HP difference per round were 96.17 when the opponent was the trained, 58.2 when it was the adapt0, and 55.3 when it was the adaptT. In addition, the winning rates of the user were 0.833 when it was the trained, 0.767 when it was the adapt0, and 0.633 when it was the adaptT. Note that the best winning rate is 0.5 (even) and the worst is 1.0 (always win) or 0.0 (always lose) from the viewpoint of adaptation.

Table 7 shows a summary of the HP differences and the winning rates for all subjects. It shows that, for at least 15 subjects, both of the proposed agents were more matching

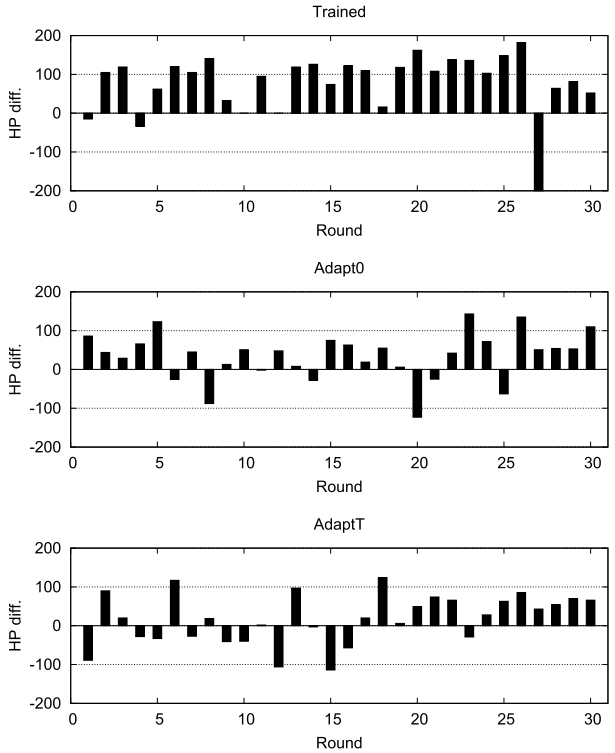

Fig. 8 The transition of final HP differences when a subject played against the `trained`, the `adapt0`, and the `adaptT`. This subject played 30 rounds with each agent. *X*-axis shows the number of rounds and *y*-axis shows the HP difference, in which a positive value shows the winning of the subject.

Table 7 Comparing the `trained`, the `adapt0`, and the `adaptT` from each subject's point of view.

(a) Average of final (absolute) HP differences

better	↔	worse	↔	worse	#subjects
trained	<	adapt0	<	adaptT	3
trained	<	adaptT	<	adapt0	3
adapt0	<	trained	<	adaptT	4
adapt0	<	adaptT	<	trained	6
adaptT	<	trained	<	adapt0	5
adaptT	<	adapt0	<	trained	9

(b) Winning rate difference from even (= 0.5)

better	↔	worse	↔	worse	#subjects
trained	<	adapt0	<	adaptT	2
trained	<	adaptT	<	adapt0	1
trained	<	adapt0	=	adaptT	2
adapt0	<	trained	<	adaptT	4
adapt0	<	adaptT	<	trained	6
adapt0	<	adaptT	=	trained	1
adaptT	<	trained	<	adapt0	0
adaptT	<	adapt0	<	trained	12
adaptT	<	adapt0	=	trained	1
adaptT	=	adapt0	<	trained	1

than the `trained`, whereas the `trained` was the best of the three for at most six subjects.

5. Discussion

Although the `strong` agent received more "most fun" qualifications, the `adaptT` agent got the less amount of negative ratings. That is, the `adaptT` satisfied the objective of this work where all users at different levels of expertise would enjoy the game. However, it is slightly regrettable that the proposed agents did not receive the best rating. We believe that the reason why the `strong` agent received better ratings than the proposed agents is that the `strong` was challenging for the users while the proposed agents tried to be a similar level to each user's. Table 4 shows that the average difficulties of the proposed agents (and the `trained`) were less than 3, which means that each user considered them slightly easy to defeat. If the reward given to MSA had been slightly larger when the agent had won, *i.e.*, the proposed agents had been designed to *defeat the users* by a small HP difference, they would be perceived as more challenging by the users and might outperform the best static opponent. Nonetheless, since the length of play was not much different between the `strong` and the proposed agents, the proposed agents provided users similar time of entertainment, even though they gave the users slightly less fun than the `strong`.

Although having an agent adapt itself by only comparing the HP difference is a very elegant idea, this method arose some problems: unnecessary actions and reinforcing suicide. The rationality theorem of Profit Sharing [7] guarantees that no ineffective rule will be part of the agent's policy, but it does not prevent unnecessary actions to be reinforced. If the agent, while exploring actions, executes kicks 15 units away from the user, and by chance the HP difference is low enough at the end of the round, then the agent will execute unnecessary kicks next time it is a 15 units from the user. Although such redundant actions might eventually disappear as the learning proceeds, they make the agent *look like* stupid. Moreover, it has been observed that, in some cases, the agent learned to lower the HP of the user to a small level, and then jump from the platform. In the game rule shown in Sect. 4, falling from the platform equals losing the round, *i.e.*, it reduces the HP of the agent to 0, which results in a small HP difference and a high reward. In order to solve these problems the reward could be assigned with more considerations than only HP difference, or the reward could be shared using a reinforcement function that discerns whether or not the rule to be reinforced was essential in attaining a particular result.

6. Related Work

Videogames including fighting videogames are the domain where artificial intelligence techniques have been used. However, the used techniques are usually traditional and static. As Graepel *et al.* [8] exemplifies, commercial fighting videogames use Finite State Machines for the design of their agents controlling the opponent characters. This traditional approach prevents the agents from learning and adapting, which means that such opponents are in a rut and users can memorize their behaviors after repetitive plays.

Nakano *et al.* [9] dealt with the problem of a static opponent being uninteresting. They proposed an agent that

learns behaviors from different users and adds them into its lists of behaviors. Hence, their agent is always creating new routines, but they do not aim adapting to the user.

There are several studies of constructing adapting machine opponents. They have two directions: constructing the strongest opponent based on the user's behavior patterns, *e.g.*, [10]; and constructing the one having the similar level of the user. The problems in the latter direction are called "game balancing". Yannakakis and Hallam [11] explored the game balancing problem in the game Pac-Man, which is of a lower complexity compared to fighting videogames.

Andrade *et al.* [12], [13] dealt with the game balancing problem in a fighting videogame using Q-learning, a reinforcement learning algorithm. To the best of our knowledge, their work is the most similar to ours, but it does not consider combos that are found in most modern fighting videogames at all. It means that the agent does not have to predict the user's behaviors because all attacks can be blocked by the block action.

The adaptation of agent in this work can be considered as a kind of human-computer interaction (HCI), specifically an adaptive user interface [14], which assists its user by predicting his/her intention through his/her model created by monitoring how he/she has used it. Barr *et al.* [15] discussed videogames from an HCI perspective. According to them, most of existing HCI studies intend to provide efficient, error-free, easy-to-learn interfaces, because they let people work quickly and result in high productivity. Instead, videogames are often desired to be difficult and to take long time to complete, which is the opposite direction from the existing HCI studies. Therefore, they argued videogame "values" that will be useful for the study of videogame interfaces.

7. Conclusion

This work proposed an artificial agent for a fighting videogame that can adapt to the user's fighting style and to the user's level, allowing users to enjoy the game even when playing alone. The adapting agent consists of three subagents: Main Subagent (MSA), Executing-Combo Subagent (ECSA) and Receiving-Combo Subagent (RCSA). MSA moves the character, executes simple attacks, and passes the control to ECSA/RCSA. ECSA executes combination attacks, or *combos*, having similar difficulty to those executed by the user. RCSA learns the fighting style of the user and tries to execute the appropriate *combo-breaker*.

We asked 30 real users to play a fighting videogame we developed. In addition to the proposed opponents (`adapt0` and `adaptT`), we used three static opponents for comparison (`weak`, `strong` and `trained`). According to the questionnaire the users filled in and the play length, the `adaptT` received the least amount of negative ratings, while many users ranked the `strong` the best. It may be because the users preferred an opponent slightly stronger than them. The `strong` was challenging for the users whereas the proposed agents were considered slightly easy to defeat.

There are several future directions. First, as mentioned above, it is regrettable that the proposed agents did not receive the best rating. If the agents had been designed not to tie but to win by a hair, they would be more challenging and might be the best for the users. Second, it is valuable to use not only the difference of health points but also other kinds of data to detect users' intentions and/or preferences. In addition to data showing how they control the character, it may be possible to use physiological signals such as electroencephalograms and electrocardiograms [16].

References

[1] S.E. Ortiz B., K. Moriyama, M. Matsumoto, K. Fukui, S. Kurihara, and M. Numao, "Road to an interesting opponent: An agent that predicts the users combination attacks in a fighting videogame," Proc. Human-Agent Interaction Symposium, Tokyo, Japan, 2009.

[2] S.E. Ortiz B., K. Moriyama, K. Fukui, S. Kurihara, and M. Numao, "Three-subagent adapting architecture for fighting videogames," Proc. 11th Pacific Rim International Conference on Artificial Intelligence (PRICAI), Lecture Notes in Artificial Intelligence 6230, pp.649–654, Springer, 2010.

[3] E. Adams, Fundamentals of Game Design, 2nd ed., New Riders, Berkeley, CA, 2009.

[4] S. Arai and K. Sycara, "Effective learning approach for planning and scheduling in multi-agent domain," Proc. 6th International Conference on Simulation of Adaptive Behavior (SAB), pp.507–516, MIT Press, 2000.

[5] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.

[6] H. Cao, N. Mamoulis, and D.W. Cheung, "Mining frequent spatio-temporal sequential patterns," Proc. 5th IEEE International Conference on Data Mining (ICDM), pp.82–89, IEEE Computer Society, 2005.

[7] K. Miyazaki, M. Yamamura, and S. Kobayashi, "On the rationality of profit sharing in reinforcement learning," Proc. 3rd International Conference on Fuzzy Logic, Neural Nets and Soft Computing, Iizuka, Japan, pp.285–288, 1994.

[8] T. Graepel, R. Herbrich, and J. Gold, "Learning to fight," Proc. 5th International Conference on Computer Games: Artificial Intelligence, Design and Education (CGAIDE), Reading, U.K., pp.193–200, 2004.

[9] A. Nakano, A. Tanaka, and J. Hoshino, "Imitating the behavior of human players in action games," Proc. 5th International Conference on Entertainment Computing (ICEC), Lecture Notes in Computer Science 4161, pp.332–335, Springer, 2006.

[10] L. Lee, Adaptive Behavior for Fighting Game Characters, Master's thesis, San Jose State University, 2005.

[11] G.N. Yannakakis and J. Hallam, "Evolving opponents for interesting interactive computer games," Proc. 8th International Conference on Simulation of Adaptive Behavior (SAB), pp.499–508, MIT Press, 2004.

[12] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Challenge-sensitive action selection: An application to game balancing," Proc. 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), pp.194–200, IEEE Computer Society, 2005.

[13] G. Andrade, G. Ramalho, A.S. Gomes, and V. Corruble, "Dynamic game balancing: An evaluation of user satisfaction," Proc. 2nd Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE), pp.3–8, AAAI Press, 2006.

[14] J. Liu, C.K. Wong, and K.K. Hui, "An adaptive user interface based on personalized learning," IEEE Intell. Syst., vol.18, pp.52–57, 2003.

[15] P. Barr, J. Noble, and R. Biddle, "Video game values: Human-

computer interaction and games," Interact. Comput., vol.19, pp.180–195, 2007.
[16] V. Vachiratamporn, R. Legaspi, K. Moriyama, and M. Numao, "Towards the design of affective survival horror games: An investigation on player affect," Proc. 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction (ACII), pp.576–581, IEEE Computer Society, 2013.

Koichi Moriyama received B.Eng., M.Eng., and D.Eng. from Tokyo Institute of Technology in 1998, 2000, and 2003, respectively. After having worked as a Research Associate at Tokyo Institute of Technology from 2003 till 2005, he is currently an Assistant Professor at the Institute of Scientific and Industrial Research, Osaka University. His research interest includes artificial intelligence, multiagent systems, game theory, and cognitive science. He is a member of the Japanese Society for Artificial Intelligence (JSAI).

Simón Enrique Ortiz Branco received the B. Computer Eng. degree from Universidad Simón Bolívar in 2007 and M.Sc. from Osaka University in 2010. He is currently working as a video game programmer for PlatinumGames Inc.

Mitsuhiro Matsumoto received B.Eng., M.Sc., and Ph.D. from Osaka University in 2006, 2008, and 2012, respectively. He is currently with Mitsubishi Electric Corporation.

Ken-ichi Fukui received M.A. from Nagoya University in 2003 and Ph.D. from Osaka University in 2010. He was a Specially Appointed Assistant Professor at the Institute of Scientific and Industrial Research (ISIR), Osaka University from 2005 to 2010. He is currently an Assistant Professor at ISIR, Osaka University from 2010. His research interest includes data mining algorithm and its environmental contribution. He is a member of JSAI, IPSJ, and the Japanese Society for Evolutionary Computation.

Satoshi Kurihara received B.E. and M.E. degrees in Computer Science from Keio Univ., Tokyo, Japan, in 1990 and 1992, respectively. In 1992, he joined the Basic Research Division, at Nippon Telegraph and Telephone Corporation (NTT). He received Ph.D. in 2000 from Keio Univ. In 2004, he joined The Graduate School of Information Science and Technology/The Institute of Scientific and Industrial Research of Osaka Univ., Osaka, Japan, as associate professor. And, from April 2013, he joined The Graduate School of Information Systems of The University of Electro-Communications, as professor. His current research interest includes multiagent systems, ubiquitous computing, and complex network research. He is a member of IPSJ (Information Processing Society of Japan), JSAI (Japan Society of Artificial Intelligence), JSSST (Japan Society of Software Science and Technology), and ACM.

Masayuki Numao is a professor in the Department of Architecture for Intelligence, the Institute of Scientific and Industrial Research, Osaka University. He received a bachelor of engineering in electrical and electronics engineering in 1982 and his Ph.D. in computer science in 1987 from Tokyo Institute of Technology. He was working in the Department of Computer Science, Tokyo Institute of Technology from 1987 to 2003, and was a visiting scholar at CSLI, Stanford University from 1989 to 1990. His research interests include Artificial Intelligence, Machine Learning, Affective Computing and Empathic Computing. He is a member of Information Processing Society of Japan, Japanese Society for Artificial Intelligence, Japanese Cognitive Science Society, Japan Society for Software Science and Technology, and the American Association for Artificial Intelligence.