Vol.2010-GI-23 No.4
2010/3/8

An Interesting Opponent for Fighting Videogames

Simón E. Ortiz B.,[†1] Koichi Moriyama,[†2]
Ken-ichi Fukui,[†2] Satoshi Kurihara[†2]
and Masayuki Numao[†2]

Industry-standard opponents in fighting videogames do not adapt, making them uninteresting in the long run. To increase the level of entertainment that can be derived from playing fighting videogames against the computer, we propose an adapting agent. In this context "adapting" is interpreted as modifying the level of the opponent to match that of the user, and learning to fight against the fighting style of the user. The proposed agent utilizes mainly Profit-Sharing and Pattern-Mining for achieving adaptation. The proposed agent was developed and evaluated against static opponents with real users.

1. Introduction

Fighting videogames are a popular genre of video-games with new titles being released every year. A fighting videogame is a simulation of hand-to-hand combat. The winner of each round is the first user that lowers the energy of the opponent to zero by means of attacks.

Fighting videogames are designed to be played by at least two users competitively. 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. But, if given the option, users may prefer to play against other users.

We assume that one of the main reasons users prefer to play against other users is that the AI found in standard videogames 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, we are assuming that the adaptability of the human player makes it interesting compared to the machine. However, the machine AI used so far is typically of a simple design, just enough to make the user feel the software is reasonably smart[1)], e.g., Finite State Machines[2)], which means that standard videogames' AI is not complex enough to learn users' patterns.

Nevertheless, learning the user behavior (or his/her *fighting style*) and adapting to it in order to defeat the user should not be the aim. An agent that behaves so, would learn to easily defeat the user. An opponent that cannot be defeated is not interesting. Therefore, our aim is to adapt to the behavior of the user and to the user's level. Here lies the novelty of our research.

2. Background

2.1 Fighting videogames

In typical fighting videogames the first player that lowers the health-points (HP) of the opponent to zero is the winner of a round. The winner of a fight is the best of several rounds. Some videogames have a time limit per round.

The set of available actions in a game is $X \cup D \cup C \cup B \cup M$. X is the set of *simple attacks*. Simple attacks deal moderate damage, e.g., *punch*, *kick*, or *special attack* (long range projectile-like attacks). D is the set of defensive actions, or *blocks*. Blocks guard the character from simple attacks. C is the set of *combos*. Combos are a predefined combination of simple attacks that deal significant damage. B is the set of *combo-breakers*. Combo-breakers are special combinations that counter-attacks a combo. Each combo $c_i \in C$ might have a different combo-breaker $b_i \in B$. When the corresponding combo-breaker is executed before the opponent finishes delivering the combo, the receiving player will not receive the extra-damage of the combo; in some games the delivering player will receive the extra-damage of the combo. M is the set of movements the players can use to navigate the stage, e.g., walk and jump. When the stage is finite, falling outside the stage usually translates into an immediate lose. Different fighting videogames vary in the details and design of the possible actions.

2.2 Related Work

The researches that are more closely related to our own are agents that adapt to the user, e.g. 3), 4). Although these agents adapt to the user, their goal is to win all the fights, i.e., they are aiming to become very difficult opponents. Adaptation to the level of the user has been explored by 5), although the game

†1 Graduate School of Information Science and Technology, Osaka University
†2 Institute of Scientific and Industrial Research, Osaka University

was Pac-Man, which is of a lower complexity compared to fighting videogames.

The problem of a static opponent being uninteresting is dealt with in 6), but they do not aim adapting to the user. Instead, their agent learns behaviors from different users and adds them into its lists of behaviors. Hence, their agent is always creating new routines.

3. Proposal

As stated above, playing a modern fighting videogame can be thought of as dealing with the following tasks: executing simple attacks, executing blocks, executing combos, executing combo-breakers when receiving a combo, and moving.

We propose an agent that adapts to the user in a fighting videogame. This agent will control the actions of the opponent character. We divided the agent in three sub-agents, each of which will be in charge of some of the mentioned tasks. Dividing the tasks among sub-agents should help the agent learn each task in less time. The sub-agents are: Main Sub-agent (henceforth MSA), Executing-Combo Sub-agent (henceforth ECSA) and Receiving-Combo Sub-agent (henceforth RCSA). Since videogames must run in real-time, all the learning is delayed until the end of each round. From the agent's point of view, one round equals one episode.

3.1 Main Sub-agent (MSA)

The MSA will be in charge of executing simple attacks, blocks, and moving. When deemed appropriate the MSA will pass the control to one of the other sub-agents. The MSA utilizes Profit-Sharing for adapting.

3.1.1 Profit-Sharing

Profit-Sharing is a method from the Reinforcement Learning field[7]. The objective of Reinforcement Learning is to produce agents that learn by trial-and-error, without any prior knowledge of the task or environment.

The function that the agent is trying to learn is called a *policy*, i.e., a function that maps states into actions. An agent using Reinforcement Learning will modify its policy, increasing the probability $P(s, a)$ of executing action a in state s, if choosing action a in state s contributes to a desirable reward. The pair (s, a) of the executed action a in state s is called a *rule*.

Profit-Sharing was originally proposed by 8). The fundamental idea behind Profit-Sharing is that all the rules that were used in an episode will receive a portion of the positive reward attained at the end of the episode. This positive reward is *shared* among all the rules according to a *reinforcement function* f.

The rules are updated as follows:

$$P_{n+1}(s_t, a_t) := P_n(s_t, a_t) + f(R_n, t, T) \qquad (1)$$

where $P(s_t, a_t)$ is the probability of choosing action a in state s at time t; n is the number of the episode; f is the reinforcement function; R_n is the reward obtained at the end of the episode n; and T is time of the last action of the episode.

3.1.2 Design of the MSA

The MSA is modeled as a Profit-Sharing agent, as described in 9). A Profit-Sharing agent is divided into the following modules:

- The *state recognizer* receives the input from the environment where the agent lives, and decides the corresponding state s_t.
- The *look-up table* contains all the rules. It receives the state s_t and looks up the pairs of actions-weights $[a_i, w_i]$ associated with this particular state.
- The *action selector* chooses one action a_t from all the available a_i depending on their weight w_i. The agent executes action a_t.
- The *episodic memory* records the pair (s_t, a_t).
- At the end of the episode, the *learner* will receive the reward R from the environment, and it will apply the *reinforcement function* to all the pairs (s_j, a_j) in the *episodic memory.*

The algorithm[9] followed by the MSA is found in **Fig. 1**. Each of the subroutines invoked in Fig. 1 is explained below.

The *state recognizer* takes readings from the current state of the videogame in order to define a Profit-Sharing state s.

The *look-up table* will return a list $[a_i, w_i]$ of pairs of the available actions a_i for state s with their associated weights w_i.

The *action selector* choses an action a_i from the list of $[a_i, w_i]$ returned by the *look-up table.* Normally, the action selection in Profit-Sharing is a random roulette over the weights of the rules, i.e., an action a_i is chosen with probability equal to its weight w_i. In this case, the weight w_i of every action has to be positive. The rewards obtained at the end of an episode are positive: ideal goal states give maximum rewards, not so good final states give small positive rewards.

```
MSA():
  while True:
    $s_t$ := StateRecognizer(state of the game);
    $[a_i, w_i]$ := Look-upTable($s$);
    $a_t$ := ActionSelector($[a_i, w_i]$);
    execute action $a_t$;
    EpisodicMemory($s_t, a_t$);
    if received reward:
      execute Learner(); exit
```

Fig. 1 Pseudo-code of the MSA

Distributing positive rewards will always increase the probability of choosing an action. This approach would not work if the agent needs to weaken the probability of actions that lead to undesirable final states. If the agent receives a negative reward as a consequence of reaching an undesirable goal state, when updating the weights of the actions we run into the possibility of lowering the weight of an action to zero (the action would never be executed again) or to a negative value which random roulettes cannot handle. Hence, instead of using random roulette, the *action selector* uses Boltzmann Action Selection[7]:

$$P(a_i) = \frac{e^{w_i/\tau}}{\sum_{j=1}^{n} e^{w_j/\tau}} \tag{2}$$

where the probability $P(a_i)$ of choosing action a_i depends on its weight w_i and the weight of each of the available actions w_j. e is the natural exponential function. The parameter τ is called *temperature*. The higher the temperature, the more random the action selection becomes, i.e., the less relevant the relative weights of the actions becomes. Similarly, the lower the temperature, the more accented the difference in the weights becomes.

Using the Boltzmann equation the *action selector* chooses one action. The available actions to the agent are those defined in the videogame in question, plus passing control of the character to ECSA or RCSA. After the decided action has been executed, or the sub-agent executes its action, the MSA resumes control of the character.

The *episodic memory* will store the state s defined by the *state recognizer* and the action a chosen by the *action selector*.

The *reinforcement function* used by the *learner* is:

$$P_{n+1}(s_t, a_t) := P_n(s_t, a_t) + R_{n+1} \cdot \gamma^{T-t} \tag{3}$$

which is a geometrically decreasing function, so it satisfies the *Rationality Theorem*[10] which guarantees rational policies for Profit-Sharing. R_n is the reward received at the end of the episode n, γ is a discount factor ($\gamma < 1$), and T is the time t of the last executed action.

At the end of an episode, the environment will give the agent a reward. While higher positive rewards are given when the difference in the final HP of the agent and the user is small, 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 to defeat it.

With this selection of rewards we are reinforcing actions that put the agent at the same level of the user. Consider also that, at the same time, we are reinforcing actions that were effective against the fighting style of the user, otherwise the HP difference would be greater.

3.2 Executing-Combo Sub-agent (ECSA)

The ECSA has the responsibility of choosing which combos execute, and executing them. The ECSA maintains a set of combos that it executes. During a fight, when the ECSA is invoked by the MSA, the ECSA will randomly select a combo from its combo set C_A. Then, it will execute the selected combo. Executing a combo is very simple: the ECSA will instruct the character to execute the actions defined for the chosen combo, one by one.

The interesting part of the ECSA is how the selection of the set $C_A \subseteq C$ of combos is done. In order for the agent as a whole to be at the same level of the user, the combos that the agent executes must also be on a level close to that of the user. Hence, the responsibility of selecting the combos to be executed by the ECSA is translated into selecting combos of a level similar to that of the user. If we consider the set of the combos used by the user, the goal of the ECSA would be to create a set of combos of similar difficulty.

A naïve solution would be to copy the set of combos of the user. But an opponent that copies your actions would be an extremely uninteresting opponent.

In order to be able to create the set C_A of combos to be used by the agent of similar difficulty to set C_U of combos used by the user, we need to have a comparison function that will allow us to order sets by their difficulty.

There are many possible comparison functions that could be used for this purpose. We use the following three: ratio of used combos, indistinguishability of combos, and entropy of combo-breakers.

Let us imagine we have two users; user U_1 uses six different combos during a fight, while user U_2 uses only two different combos. It is logical to believe that user U_1 has a higher level than user U_2. A better user would execute a wider variety of combos because it would make it difficult for the opponent to predict the combo-breakers. Consequently, the ratio of used combos shown in Eq. 4 is a valid metric of the difficulty of a set.

$$\textit{used ratio}(C_E) = \frac{|C_E|}{|C|} \tag{4}$$

where C is the set of available combos for the game in question, and $C_E \subseteq C$ is a set of combos.

For sake of example, let us consider combos c_i each of which is made up of four attacks x_j from the set of simple attacks X. Let us use again our imaginary users U_1 and U_2. Let us assume that the set of combos of user U_1 is $\{x_1x_1x_1x_1, x_1x_1x_1x_2, x_1x_1x_1x_3\}$, and the set of combos of user U_2 is $\{x_1x_1x_1x_1, x_2x_2x_2x_2, x_1x_2x_3x_4\}$.

In this case, the set of user U_1 is more difficult than the set of user U_2. Since the combo-breaker must be executed *before* the last action of the combo, an opponent fighting against opponent U_1 would have to decide which combo-breaker to use when presented with the partial sequence $x_1x_1x_1\_$. Given the set of U_1, and the given initial actions, it is impossible to decide the proper combo-breaker. In the case of user U_2, the opponent would have enough information before the last action to decide the combo-breaker. Hence, a set of combos where the combos are indistinguishable given the initial actions, is a more difficult set than a set where the combos can be distinguished by their initial actions. The equation for the indistinguishability metric is:

$$\textit{indistinguishability}(C_E) = \frac{\sum |\textit{combos with repeated initial actions in } C_E|}{|C_E|} \tag{5}$$

where $C_E \subseteq C$ is a set of combos.

Let us imagine that all the combos c_k found in the set of combos of user U_1 have a different combo-breaker b_k, while all the combos c_l in the the set of combos of user U_2 share the same combo-breaker b_1. In that case, the combo-breakers for user U_1 would be $\{b_1, b_2, \ldots, b_n\}$, and that of user U_2 would be $\{b_1, b_1, \ldots, b_1\}$. An opponent playing against user U_2 would not need to decide combo-breakers when confronted with a combo. It is enough to always select b_1. On the other hand, an opponent playing against user U_1 would have to select a different combo-breaker when present with different combos. Evidently, the set of user U_2 is easier than that of user U_1. The characteristic we are looking at is the entropy of the set of combo-breakers:

$$\textit{breaker entropy}(C_E) = \frac{-\sum_{i=0}^{N} P(b_i)\log P(b_i)}{\log N} \tag{6}$$

where N is the number of distinct combo-breakers the combo set $C_E \subseteq C$ contains, log is the natural logarithm, and $P(b_i)$ is the probability of randomly choosing the combo-breaker b_i out of the combo-breakers of C_E.

The first time the ECSA is executed it will create a combo set containing m combos. In this set the combo-breakers of all the combos are different, and the initial actions of all the combos are different (high combo-breaker entropy, low indistinguishability). We consider that such an initial set is not too difficult, but it is not too easy either.

After finishing an episode, this combo set will be *partially* adapted to that of the user. For adapting the combo set we use implicitly *one* iteration of Hill Climbing[11)] by greedily choosing the first option that gets one of the metrics closer to its goal.

For adapting its set, the ECSA follows the algorithm presented in **Fig. 2**. C_U is the set of combos of the user, C_A is the set of combos of the agent, the parameter Δ defines the level of tolarence in the difference of sizes of the sets,

```
adaptECSA():
  if used ratio ($C_A$) > used ratio($C_U$)+$\Delta$: delete($C_A$)
  elif used ratio ($C_A$) < used ratio($C_U$)$-\Delta$: add($C_A$)
  else: swap($C_A$)
delete($C_A$):
  for($i := 0$; $i <$ top; $i := i + 1$):
    $c :=$ random combo from $C_A$;
    if $|\mathrm{entr}(C_A - c) - \mathrm{entr}(C_U)| < |\mathrm{entr}(C_A) - \mathrm{entr}(C_U)|$ or
       $|\mathrm{inds}(C_A - c) - \mathrm{inds}(C_U)| < |\mathrm{inds}(C_A) - \mathrm{inds}(C_U)|$:
      $C_A := C_A - c$; exit
```

Fig. 2 Pseudo-code of the ECSA

and the parameter top is a limit imposed on the number of tries that will be executed; this is to prevent the ECSA from cycling indefinetily if there is not a suitable combo to choose. The subroutine `delete` is presented along with the algorithm. The subroutines `add` and `swap` follow the same idea as `delete`. The metrics *entr* (entropy) and *inds* (indistinguishability) are defined as explained above.

3.3 Receiving-Combo Sub-agent

The design of the RCSA is presented extensively in 12). This sub-agent basically mines the patterns with which the user executes combos. The mining occurs after each episode. For pattern mining the RCSA uses Substring Tree[13].

When the RCSA executes during a round, it matches the combos executed by the user with mined patterns; using the matched patterns, the RCSA predicts the next possible combos. Then the RCSA chooses the combo-breaker stochastically based on the relative frequency of the predicted combos. For more details of this sub-agent see 12).

4. Experiments

We developed a simple fighting videogame to test the proposed adapting agent. Our videogame was developed using the Open Source game engine Crystal Space 3D[14]. An image of the videogame can be seen in **Fig. 3**.

In order to evaluate whether the proposed agent is more entertaining than static agents, we asked 28 real users to play it. The users were of different nationalities, ages and with different level of expertise at playing videogames.

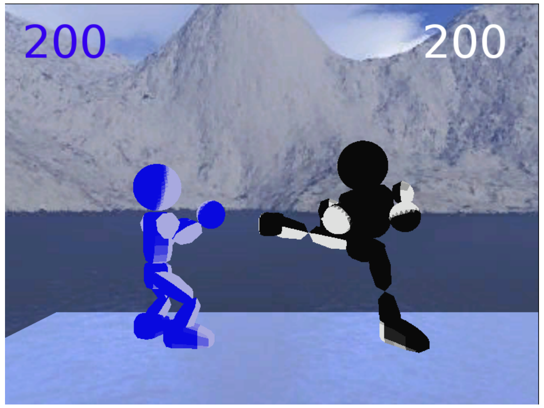

Fig. 3 An screen capture of our simple fighting videogame

The fighting videogame developed 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 fight; there is no time limit; there is one round per fight; the initial HP of the characters is 200; the set X of simple attacks contains punch (`p`), kick (`k`) and special attack (`s`), the last one being a long range projectile attack; each of the simple attacks deal 1 point of damage; the set D of defenses contains one action: block; while blocking, simple attacks do not have effect; the set C of combos 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; in order for the combo-breaker to be valid it must be executed before the opponent completes the combo; if the combo-breaker is valid, the character executing the combo will receive its damage; the set M of movements contains move to the right, move to the left, jump and crouch.

The proposed agent was used with the following parameters:

- For the MSA: The discount factor γ of the *reinforcement function* is fixed at 0.99. The *temperature* parameter τ of the Boltzmann action selection is

Table 1 List of Combos

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

fixed at 1.0

- For the *state recognizer* of the MSA: In order to keep the design of the agent simple, we discretized the world state as explained below.
  - Is the agent/user crouching?
  - Is the agent/user jumping?
  - Is the agent receiving a combo?
  - Is the user executing a simple attack?
  - Is the agent/user blocking?
  - Is the agent near a border of the stage?
  - Is the user in danger of losing (HP below 15%)?
  - 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.
  - 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.
  - The difference in HP between the user and the agent, rounded to tens.

  These characteristics were selected to define a state s because they provide enough information to the agent to make intelligent decisions.
- For the *action selector* of the MSA: the available actions are those available in the game, plus ECSA, RCSA, and stay. Instead of the actions right and left, the agent uses approach and withdraw, which are independent of the character changing the direction to which it faces. Each of the actions approach, withdraw, crouch and block are executed for 0.1 seconds. The action stay has a duration of 0.4 seconds. All the other actions last as long as it takes to fully execute them.
- For the RCSA: The number of patterns that are being track simultaneously is five.

Table 2 Assignment of 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

- For the ECSA: The number of combos in the original set is three, the parameter Δ is 0.1, and the parameter top is 20.

The reward given to the agent by the environment after one episode is defined in **Table 2**. The HP difference is the absolute difference between the HP of the user and the agent.

For comparison purposes we developed three static agents: `weak`, `medium` and `strong`. The `weak` agent is very easy to defeat, 50% of its action are to wait; it only executes combos 0 and 10 from Table 1; the combo-breaker it uses is always `p`. The `medium` agent was obtained by training our adapting agent against a user for 20 rounds; while the `medium` agent is being used for the evaluation it will not adapt. The `strong` agent is very difficult to defeat; it will always get close to the opponent, it will use combos whenever close enough; the combos used are all the available combos, executed in a random order; the combo-breakers it uses are chosen stochastically based on the distribution of combo-breakers for the initial actions executed by the user (e.g., if the user executes `ksp_` the `strong` agent will execute either `p` or `s` with 50% probability each, if the user executes `kkk_` the `strong` agent will execute `k`).

We compared these static agents against two versions of our adapting agent: `adap0` and `adapF`. The `adap0` agent is as explained in Section 3. The `adapF` agent is structurally the same as `adap0`, but it has been trained by playing 20 rounds beforehand. In comparison with `adap0`, `adapF` already knows certain rules from the environment, such as falling from the platform equals finishing the round, or that executing combo-breakers when receiving combos lowers the damage of the opponent, etc.

We asked the users to play between 15 and 30 rounds against each agent. After the first 15 rounds with an agent, the user could quit whenever he/she was no longer having fun. The users did not know the characteristics of each agent.

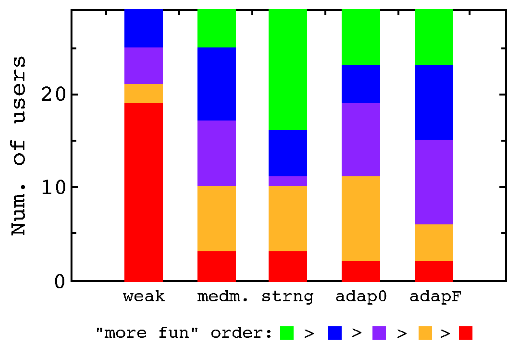

Fig. 4 Questionnaire results

The order of the agents was randomized for the users. The users filled in a questionnaire after the experiments. They were asked to order the agents from most fun to least fun.

5. Results

The results of the survey are shown in **Fig. 4**. The opponent that appeared as the most fun in the survey of a test subject was assigned one green point, the second of the list was assigned a blue point, etc. Although the `strong` agent received more "most fun" qualifications, the proposed agent `adapF` got the less amount of negative ratings.

We also compare the length of play against each opponent. Similar to the survey, the opponent that was played the most received one green point, the second opponent a blue point, etc. In case of draws, both opponents received the point corresponding to their position. The comparison is shown in **Fig. 5**. Similar to the survey, the proposed agents were the opponents that figured less in the least played opponents.

The results of each round for one of the test subjects can be seen in **Fig. 6**. The behavior of the results do not change drastically for other subjects. Whether the proposed agents is in fact adapting to the user or not, cannot be deduced from these graphics. The rapid variations in the HP results of the proposed agents could be consequence of the agent compensating its own difficulty, and/or

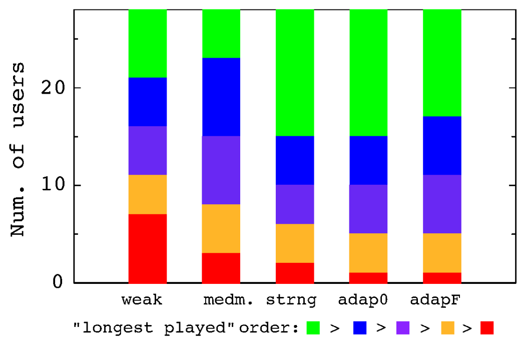

Fig. 5 Analysis of playing length

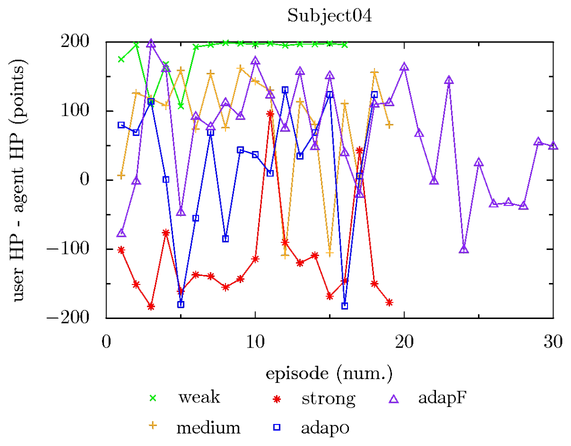

Fig. 6 HP differences for one subject

consequence of the users trying new strategies.

6. Discussion

We believe that the reason why the `strong` agent received better ratings than the proposed agent, is that the proposed agent's goal is to have a difficulty similar to that of the user. If the goal were to defeat the user *by a certain HP difference*, the proposed agent would be perceived as more challenging by the users and might outperform the best static opponent.

Although the `strong` opponent received better ratings, the length of play-time is not much different than that of the proposed agents. Although not considered as fun as the `strong` agent, the proposed agents provide similar time of entertainment.

Given that the goal of the agent is to defeat the user by a certain amount of HP, could the proposed agent be used *as is* in an industry videogame? The authors are of the opinion that this is not the case yet. Although having an agent that adapts 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[10] 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. The accumulation of unnecessary actions in the policy does not make the agent look like a smart opponent.

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. Falling from the platform reduces the HP of the agent to 0, which becomes 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 function that discerns whether or not the rule to be reinforced was essential in attaining a particular result.

7. Conclusions

An agent that adapts to the user's fighting style and to the user's level was developed. The adapting agent is divided in three sub-agent: MSA, ECSA and RCSA, each of which is in charge of handling different aspects of fighting. In comparison with static opponents the adapting agent received the least amount of negative ratings. We believe that the reason why the `strong` agent received better ratings is that the proposed agent's goal is to have a difficulty similar to that of the user. If the goal were to defeat the user by a certain HP difference, the proposed agent would be perceived as more challenging by the users and might outperform the best static opponent.

References

1) Adams, E.: *Fundamentals of Game Design*, New Riders, Berkeley, CA, 2nd edition (2009).

2) Graepel, T., Herbrich, R. and Gold, J.: Learning to fight, *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pp.193–200 (2004).

3) Lee, L.: Adaptive Behavior for Fighting Game Characters, Master's thesis, San Jose State University (2005).

4) Ricciardi, A. and Thill, P.: Adaptive AI for Fighting Games. Final project of CS 229 Machine Learning, Standford University (2008).

5) Yannakakis, G. and Hallam, J.: Evolving Opponents for Interesting Interactive Computer Games, *Proceedings of the Eighth International Conference on the Simulation of Adaptive Behavior; From Animals to Animats 8*, pp.499–508 (2004).

6) Nakano, A., Tanaka, A. and Hoshino, J.: *Imitating the Behavior of Human Players in Action Games*, Lecture Notes in Computer Science, Vol.4161, pp.332–335, Springer, Berlin / Heidelberg (2006).

7) Sutton, R.S. and Barto, A.G.: *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, MA (1998).

8) Grefenstette, J.J.: Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms, *Mach. Learn.*, Vol.3, No.2–3, pp.225–245 (1988).

9) Arai, S. and Sycara, K.: Effective Learning Approach for Planning and Scheduling in Multi-Agent Domain, *Proceedings of the Sixth International Conference on the Simulation of Adaptive Behavior; From Animals to Animats 6*, pp.507–516 (2000).

10) Miyazaki, K., Yamamura, M. and Kobayashi, S.: On the Rationality of Profit Sharing in Reinforcement Learning, *Proceedings of the 3rd International Conference on Fuzzy Logic, Neural Nets and Soft Computing*, pp.285–288 (1994).

11) Russell, S. and Norvig, P.: *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, 2nd edition (2003).

12) Ortiz B., S.E., Moriyama, K., Matsumoto, M., Fukui, K., Kurihara, S. and Numao, M.: Road to an Interesting Opponent: An Agent that Predicts the Users Combination Attacks in a Fighting Videogame, *Proceedings of the Human-Agent Interaction Symposium* (2009).

13) Cao, H., Mamoulis, N. and Cheung, D. W.: Mining Frequent Spatio-Temporal Sequential Patterns, *Proceedings of the Fifth IEEE International Conference on Data Mining*, pp.82–89 (2005).

14) `http://www.crystalspace3d.org/`: Crystal Space 3D.