Proc. of 2014 IEEE Conference on Computational Intelligence and Games (CIG 2014), Dortmund, Germany, pp. 437-441, Aug. 26-29, 2014.

Deduction of Fighting-Game Countermeasures Using the k-Nearest Neighbor Algorithm and a Game Simulator

Kaito Yamamoto
Intelligent Computer Entertainment Laboratory
Ritsumeikan University
Shiga, Japan

Syunsuke Mizuno[1]
Intelligent Computer Entertainment Laboratory
Ritsumeikan University
Shiga, Japan

Chun Yin Chu
Department of Computer Science and Engineering
Hong Kong University of Science & Technology
Hong Kong, PR China

Ruck Thawonmas
Intelligent Computer Entertainment Laboratory
Ritsumeikan University
Shiga, Japan
ruck@ci.ritsumei.ac.jp

***Abstract*—This paper proposes an artificial intelligence algorithm that uses the k-nearest neighbor algorithm to predict its opponent's attack action and a game simulator to deduce a countermeasure action for controlling an in-game character in a fighting game. This AI algorithm (AI) aims at achieving good results in the fighting-game AI competition having been organized by our laboratory since 2013. It is also a sample AI, called MizunoAI, publicly available for the 2014 competition at CIG 2014. In fighting games, every action is either advantageous or disadvantageous against another. By predicting its opponent's next action, our AI can devise a countermeasure which is advantageous against that action, leading to higher scores in the game. The effectiveness of the proposed AI is confirmed by the results of matches against the top-three AI entries of the 2013 competition.**

I. Introduction

A fighting game is a genre of game in which humanoid or quasi-humanoid characters, individually controlled by normally two players, engage in a hand-to-hand combat or a combat with melee weapons, and the winner is determined by comparing the the amount of damages taken by each side within a limited time. Gameplay styles of fighting games include PvP-Game, in which a human player fights against another human player, and Versus-AI-Game, where a human player fights against a character controlled by artificial intelligence algorithms (AIs). Nowadays, the mainstream gameplay style of fighting games is PvP-Game, and Versus-AI-Game in fighting games is usually regarded by players as the gameplay for practices of game control.

However, most of the existing AIs are rule-based ones, where their actions are merely determined by various attributes of the game, such as the characters' coordinates or damage amounts. Such rule-based actions may unwittingly lead to their AI being hit by the player and thus taking damages. As a rule-based AI repeatedly employs the same pattern, it will employ the same action, even if that tactic has been proven ineffective against the player, whenever the same condition arises. Thus, if its opponent player intentionally reproduces the same condition, the AI will repeatedly employ the same ineffective tactic.

[1]The author has joined Dimps Corporation since April 2014.

Fig. 1. Screen-shot of FightingICE where both sides use the same character from the 2013 competition.

To avoid such situations, an AI must be able to choose from a variety of action patterns. As such, we can derive that a fighting-game AI, aimed at being a good practice partner for human players, should be able to formulate tactics advantageous to itself [1]–[3], without relying on a definite set of rules which is often prone to manipulation by its opponent human-player. This paper utilizes the FightingICE platform[1] and solves the aforementioned issue in existing fighting-game AIs by proposing an AI that predicts its opponent's next attack action with the k-nearest neighbor algorithm [4] and deduces the most reasonable countermeasure action accordingly.

II. FightingICE and its Competition

Since 2013, our laboratory has been organizing a game AI competition using the aforementioned FightingICE, a 2D fighting-game platform [5] for research purposes. FightingICE is a fighting game where two characters engage in a one-on-one battle (Fig. 1). In this game, each character is given numerous actions for performing at its disposal, and, as a restriction, it is not notified of the latest game information forthwith, but after a delay of 0.25 second.

[1]http://www.ice.ci.ritsumei.ac.jp/˜ftgaic/

A. Game Rules

In FightingICE, $\frac{1}{60}$ second represents 1 frame, and the game progress is advanced at every 1 frame. A 60-second (3600-frame) fight is considered as 1 round, and the winner is decided on the scores, defined below, obtained from 3 rounds, comprising 1 match. In a round, there is no upper limit to the amount of damages, and the damages inflicted on both characters are counted towards the scores as follows, where the final amount of inflicted damages at a given round is represented as HP, standing for hit points. Let the HP of the character of interest and that of the opponent character be $selfHP$ and $opponentHP$, respectively. The former character's scores in the round of interest are calculated by the following formula.

$$scores = \frac{opponentHP}{selfHP + opponentHP} \cdot 1000 \quad (1)$$

If both sides are taking the same amount of damages, each of them will be granted 500 scores. The goal in a given match is to compete for a larger share of the sum of the total scores for the three rounds in the match.

B. Character

Among four characters available in the final version of FightingICE for the 2014 competition, we used a character called KFM[2], which was the only available character in the 2013 competition. This was done so that we could fairly evaluate the performance of our AI against the top-three AIs in the 2013 competition in Section IV. Thereby, the character of each side has exactly the same ability and is controlled using seven input keys: UP, DOWN, LEFT, RIGHT, A, B, and C. In addition, there are two action types: active actions in response to the player's or AI's input and passive actions in response to the game system. All active actions are categorized into three categories: movement, attack and defense, details of which are described below.

1) Movement: A movement is an action that changes the character's position. The character can move left or right, and jump into the air. Every movement has a preset scale, by which the character's coordinates are changed. A movement can only be performed while the character is on ground.

2) Attack: An attack is an action that generates an attack object. The attack category are further classified into four types: high-attack, middle-attack, low-attack and throw-attack. These four types are related to the later-mentioned defense actions. When an attack object generated by an attack action coincides with the opponent, the opponent receives a damage, and the attack object then disappears. Every attack undergoes three states (Fig. 2): startUp, active and recovery, described as follows:

StartUp	The state between the start of the action and the generation of an attack object.
Active	The state between the generation of the attack object and its disappearance.
Recovery	The state between the disappearance of the attack object and the character's readiness for the next action.

[2]KFM is, however, not an official character for the 2014 competition. It is included in the 2014 release so that all AIs entries for the 2013 competition, available in the competition site, can also be tested on the 2014 platform.

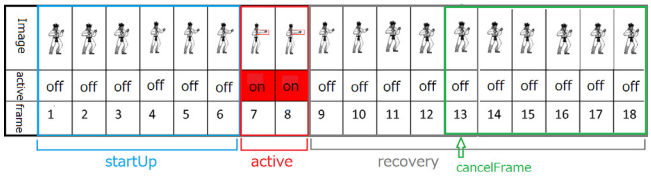

Fig. 2. Three states of an attack action.

Note that a character cannot perform any other action during execution of an attack action. As an exception, after the attack object hits the opponent, the character can cancel the current attack action and perform another action within a period called cancelFrame, which is individually defined for each attack action.

3) Defense: A defense is an action that minimizes the amount of damages inflicted by the opponent. There are three defense types, stand-defense, crouch-defense, and air-defense, each of which is effective against different attack types. A high-attack can be guarded against by a stand-defense, crouch-defense, or air-defense; a middle-attack can be diverted by a stand-defense or air-defense; a low-attack can be dodged by a crouch-defense only. A damage caused by a throw-attack cannot be mitigated by any kind of defense, but can be avoided by staying in the air.

C. AI Creation Rules in the Competition

AI creation rules are based on those used in the Ms Pac-Man vs Ghosts League Competition [6]. They are as follows:

- Initialization time is 5 seconds
- Memory usage is limited to 512MB
- Usage of multithread is forbidden
- Maximum size of file reading/writing is 10MB
- Any conduct deemed fraudulent is prohibited

III. Proposed Methodology

This paper attempts to solve the issue residing in rule-based fighting-game AIs, discussed in Section I, by proposing an AI which can predict the opponent's next attack action and devise an effective countermeasure against the predicted attack action. In order to do this task, from the start of a match, our AI records all of the attack actions by the opponent on relative coordinates between the player and the opponent (hereafter, relative coordinates). The next attack action most likely to be taken by the opponent is then predicted by a representative attack action among those conducted so far by the opponent near the current relative coordinates. The approach adopted for this prediction task is pattern classification of the opponent's attack action by the k-nearest neighbor algorithm.

A. Data Collection

Because the opponent's attack patterns vary for every opposing player, we have to collect data in real-time. Data collection is conducted using Algorithm 1.

Algorithm 1 collectData($self$, $opponent$, $data$)

```
if opponent.act is an attack action then
  x ⇐ opponent.x − self.x
  if self is not facing to the right then
    x ⇐ −x
  end if
  y ⇐ self.y − opponent.y
  position ⇐ checkPosition(self, opponent)
  if position is ground − ground then
    data.gg.add(opponent.act, x, y)
  else if position is ground − air then
    data.ga.add(opponent.act, x, y)
  else if position is air − ground then
    data.ag.add(opponent.act, x, y)
  else if position is air − air then
    data.aa.add(opponent.act, x, y)
  end if
end if
```

In this algorithm, $self$ and $opponent$ stand for our AI's character and the opponent's, respectively. The variables $self.x$, $self.y$, $opponent.x$ and $opponent.y$ represent the characters' absolute coordinates while x, y are the relative coordinates. The variable $opponent.act$ denotes the action the opponent is currently performing. The variable $data$ means the union of all sets of data, within which the data set for each of the four positions, gg, ga, ag and aa, described below, resides.

At the outset of each action by the opponent, the algorithm judges whether the action being performed is an attack or not. If it is an attack, the algorithm will acquire the type of attack and the current relative coordinates. The absolute coordinate origin for the in-game values of $self.x$, $self.y$, $opponent.x$ and $opponent.y$ is set at the upper left corner. Thereby, the variables $self.x$ and $opponent.x$ increase as the corresponding character moves to the right while $self.y$ and $opponent.y$ increase as the corresponding character moves towards the bottom.

It should be noted that when collecting attack data on relative coordinates, our character's position is regarded as the coordinate origin. In addition, the plus direction for the x value is the direction the character is facing, and the plus direction for the y value is upward. Using $checkPosition(self, opponent)$, the algorithm determines the current positions of both characters. Such positions are classified into the following four categories:

- Both characters are on ground ($ground - ground$)
- Our character is on ground while the opponent's is in the air ($ground - air$)
- Our character is in the air while the opponent's is on ground ($air - ground$)
- Both characters are in the air ($air - air$)

Using $add(opponent.act, x, y)$, the algorithm adds the opponent's current attack data, consisting of the current attack action and relative coordinates, into the data set corresponding to the current positions of both characters.

Algorithm 2 decideAction($self$, $opponent$, $data$, $distThreshold$, $numAct$, k, $game$)

```
x ⇐ opponent.x − self.x
if self is not facing to the right then
  x ⇐ −x
end if
y ⇐ self.y − opponent.y
position ⇐ checkPosition(self, opponent)
if position is ground − ground then
  actData ⇐ data.gg
else if position is ground − air then
  actData ⇐ data.ga
else if position is air − ground then
  actData ⇐ data.ag
else if position is air − air then
  actData ⇐ data.aa
end if
count ⇐ 0
for i = 1 to actData.num do
  distance ⇐ calculateDist(actData[i], x, y)
  if distance < distThreshold then
    count + +
  end if
end for
if count >= numAct then
  predictAct ⇐ kNN(k, actData, x, y)
  self.playAct
  ⇐ simulate(self, opponent, predictAct, game)
else
  self.playAct ⇐ guardAction
end if
```

B. Action Decision

When the opponent is about to perform an attack action, our AI compares the relative coordinates at that time with the opponent's attack data collected hitherto and it predicts which attack action the opponent is going to perform. First, the AI predicts whether its opponent is going to perform an attack or not. To do this, the AI finds the current relative coordinates. If there are a certain number of the opponent's past attack actions conducted within a predefined area around the current relative coordinates, the AI will judge that the opponent is going to perform an attack and classify the opponent's attack action using the k-nearest neighbor algorithm. The AI then decides its own action based on the result from a game simulator. The details of this decision making are given in Algorithm 2.

In this algorithm, $actData.num$ is the number of data recorded in the data set of the corresponding positions of both characters. The parameter $distThreshold$ is the distance threshold used in judging whether the opponent is going to perform an attack action or not; it will be judged that the opponent is going to attack if the number of nearby past attack actions – with the distance from the current relative coordinates less than $distThreshold$ – exceeds the threshold value $numAct$. The parameter k is a reference number of neighbor data to be used in judging which action will be performed by the opponent while $guardAction$ represents the default defense action of the character: CROUCH_GUARD.

At first, this algorithm collects the current relative co-

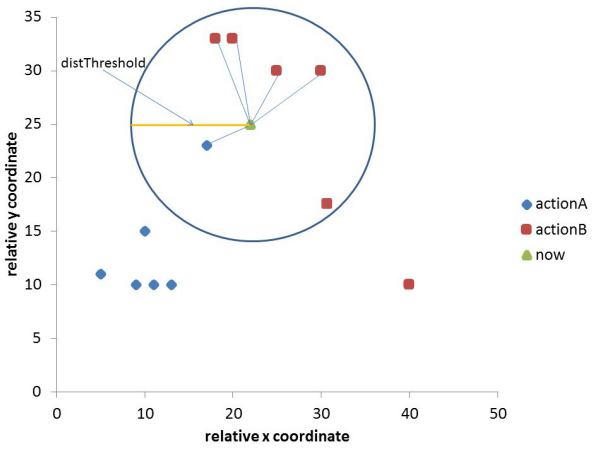

Fig. 3. Prediction of the opponent's next attack action with the k-nearest neighbor algorithm ($k = 5$).

ordinates and both characters' current positions. And then, the algorithm selects the data set which corresponds to those positions. For each data in the selected data set, the distance from the current relative coordinates is calculated using $calculateDist(actData[i], x, y)$. If the number of data whose distance is less than $distThreshold$ is $numAct$ and above, the algorithm will start action prediction. Applying the k-nearest neighbor algorithm to the action data set and the current relative coordinates, the algorithm uses $kNN(k, actData, x, y)$ to extract the type of attack action which bears the highest number of occurrences in the k nearest data around the current relative coordinates (Fig. 3). If there are several such action types, all of them will be extracted. Several values of the parameter k are examined in Section IV.

In the example shown in Fig. 3, the value of k is set to 5. The point "now" is the current relative coordinates while all the other points are the relative coordinates of the opponent's previous actions whose shape represents their action type. In this example, there are two action types: actionA and actionB. The circle encircling "now" represents the area within the range of $distThreshold$ and contains six previous data points. Those five data points connected to "now" by lines are the neighbor data points identified by the k-nearest neighbor algorithm, i.e., one actionA and four actionB's; according to majority voting adopted therein, actionB is thus extracted. The extracted action is passed to $simulate(self, opponent, predictAct, game)$, which simulates all possible countermeasures and calculates their evaluation values and then decides the next action of our AI's character. The details of the simulator are described in the coming section.

C. Simulator

The simulator is incorporated within the AI and conducts simulation with all combinations of the opponent's predicted attack actions and each of the actions which can be performed by our AI. For each combination, the simulator simulates the game up to one second from the current time. The AI then chooses the action with the highest evaluation value as its next action. The evaluation value for each action of the AI is determined by the amount of damages of the opponent minus that of the AI. The details of this are given in Algorithm 3.

Algorithm 3 $simulate(self, opponent, predictAct, game)$

```
for i = 1 to self.action.size do
  E[i] ⇐ 0
  for j = 1 to predictAct.size do
    fight(self.data, opponent.data, game.data)
    for k = 1 to 60 do
      updateCharacter()
      if self is controllable then
        self.act ⇐ self.action[i]
      end if
      if opponent is controllable then
        opponent.act ⇐ predictAct[j]
      end if
      calculateAttackParameter()
      calculateHit()
    end for
    E[i] ⇐ E[i] + opponent.damage − self.damage
  end for
end for
iMax ⇐ arg max E
if E[iMax] < 0 then
  return guardAction
end if
return self.action[iMax]
```

In this algorithm, $action.size$ is the number of active actions which can be performed by our AI. Due to restricted computation time, instead of using all active actions available in FightingICE, the simulator only considers 24 typical active actions, i.e., 16 on-ground actions and 8 air actions, listed in Table I. In addition, $predictAct.size$ is the number of the opponent's predicted attack actions from Algorithm 2 while $game.data$ represents the aggregation of all data used in the game. The variables $self.damage$ and $opponent.damage$ are the amount of damages of the AI's character and that of the opponent's character in the simulation, respectively.

First of all, the AI calls all 24 active actions. Using brute force, the AI conducts simulation of each called action with each of the opponent's predicted actions. The information of both characters and that of the game are input into the simulator through $fight(self.data, opponent.data, game.data)$. Then, the simulator executes those two actions, when they can be performed, or controllable, by their character, for the period simulating the next 60 frames or 1 second.

Any changes to each character in each frame are applied in $updateCharacter()$. For each character, when it is ready for an action, the selected action is performed. Any changes to all issued attack objects in each frame are applied in $calculateAttackParameter()$. The function $calculateHit()$ identifies a collision between an attack object and its target character and handles the necessary process following the collision. The three functions $updateCharacter()$, $calculateAttackParameter()$ and $calculateHit()$ reuse similar functions available in the Fighting class of the main FightingICE program.

After 60 frames, the algorithm calculates the difference between the amount of damages inflicted on the opponent's character and that on the AI's character, which is taken as the evaluation value for the action performed by the AI. After all

TABLE I. LIST OF ACTIONS USED IN THE SIMULATOR

JUMP	FOR_JUMP
BACK_JUMP	THROW_A
THROW_B	STAND_A
CROUCH_A	STAND_FA
CROUCH_FA	STAND_D_DF_FA
STAND_D_DF_FB	STAND_F_D_DFA
STAND_F_D_DFB	STAND_D_DB_BA
STAND_D_DB_BB	STAND_D_DF_FC
AIR_GUARD	AIR_A
AIR_DA	AIR_FA
AIR_UA	AIR_D_DF_FA
AIR_F_D_DFA	AIR_D_DB_BA

TABLE II. AVERAGE SCORES AGAINST T (2013 WINNER) FOR 100 MATCHES

k	Round 1	Round 2	Round 3	Total
1	513.6	598.4	502.8	1614.8
3	571.1	556.5	512.7	1640.2
5	584.4	542.1	532.2	1658.6
7	566.8	639.3	571.3	1777.5
9	556.5	657.7	602.0	1816.2
11	496.0	617.2	600.3	1713.4

combinations have been simulated, the algorithm returns the action with the highest evaluation value as its output.

IV. PERFORMANCE EVALUATION

Performance evaluation was done by matching the proposed AI with the top three AI entries of the 2013 competition: T (the winner), SejongAI (the runner-up), and Kaiju (the 3rd place). We used the latest version of FightingICE for the 2014 competition and used the character KFM for both sides. Since we wanted to examine the effect of k in the k-nearest neighbor algorithm, k was regarded as a variable in this evaluation. The value of k was set to 1, 3, 5, 7, 9 and 11. For each value of k, the AI played 100 matches with each opponent, and its average scores were recorded. Parameters other than k were set as follows: $distThreshold = 40$, $numAct = k$.

Tables II - IV list the average scores of each round and the average total scores for each value of k. As the two sides were competing for the maximum total scores of 3000 in a match, one side could be said to have earned higher scores than the other if it had earned more than 1500 scores.

The results of the performance evaluation showed that the proposed AI was able to earn more average total scores than all of its opponents. For Round 1, our AI lost to T and Kaiju for $k = 11$ and $k = 5, 9, 11$, respectively, but it could recover and outperform these two opponents in the subsequent rounds. This indicates future work on how to cope with Round 1 where the amount of the opponent's recorded data is insufficient.

The best value of k, in terms of the average total scores is 9, 11 and 3 for T, SejongAI and Kaiju, respectively. For each opponent, this value of k is also the best k for Rounds 2 and 3, but the best k for Round 1 is smaller, i.e., 5, 9 and 1, for T, SejongAI and Kaiju, respectively. We, therefore, deduce that the best value of k is different for different opponents who have different tendencies in their behaviors. In addition, by switching the value of k appropriately in each round, it would be possible for the proposed AI to achieve higher scores.

TABLE III. AVERAGE SCORES AGAINST SEJONGAI (2013 RUNNER-UP) FOR 100 MATCHES

k	Round 1	Round 2	Round 3	Total
1	515.1	533.4	565.6	1614.1
3	546.9	574.3	584.1	1705.2
5	575.4	579.2	596.9	1751.6
7	594.5	622.4	578.6	1795.5
9	621.1	629.5	567.1	1817.7
11	614.7	651.5	602.4	1868.6

TABLE IV. AVERAGE SCORES AGAINST KAIJU (2013 3RD PLACE) FOR 100 MATCHES

k	Round 1	Round 2	Round 3	Total
1	618.2	623.9	633.3	1875.4
3	539.0	677.8	672.2	1889.0
5	492.7	640.6	650.4	1783.7
7	526.4	604.7	632.5	1763.6
9	459.8	570.2	628.1	1658.1
11	495.1	537.6	631.4	1664.1

V. CONCLUSIONS AND FUTURE WORK

The method proposed in this paper works effectively against the top three AI entries in the 2013 computation. Hence, predicting the opponent's attack action and devising a countermeasure accordingly is an effective approach for designing a strong fighting-game AI. However, the current version of our AI starts with the null sets of $data.gg$, $data.ga$, $data.ag$, and $data.aa$, and it collects data throughout a given match. As such, the AI suffers from inaccurate prediction of the opponent's attack actions before sufficient data have been accumulated. To cope with this issue, a rule-based algorithm could be used to guide the AI's actions during such a period. We have also found that the best k varies for different opponents and rounds. Hence, as our future work, we plan to focus on a mechanism for switching k to its effective value by analyzing the behaviors and tendencies of the opponents.

ACKNOWLEDGMENT

Game resources of FightingICE are from The Rumble Fish 2 with the courtesy of Dimps Corporation.

REFERENCES

[1] B.H. Cho, S.H. Jung, Y.R. Seong, and H.R. Oh, "Exploiting Intelligence in Fighting Action Games using Neural Networks," IEICE Transactions on Information and Systems, vol. E89-D, no. 3, pp. 1249–1256, 2006.

[2] S, Lueangrueangroj and V. Kotrajaras, "Real-time Imitation based Learning for Commercial Fighting Games," Proc. of Computer Games, Multimedia and Allied Technology 09, International Conference and Industry Symposium on Computer Games, Animation, Multimedia, IPTV, Edutainment and IT Security, pp. 1–3, 2009.

[3] S.S. Saini, C.W. Dawson, and P.W.H. Chung, "Mimicking Player Strategies in Fighting Games," Proc. of the 2011 IEEE International Games Innovation Conference (IGIC), pp. 44–47, 2011.

[4] A. Smola and S.V.N. Vishwanathan, Introduction to Machine Learning, Second Edition, pp. 168-170, The MIT Press, 2009.

[5] F. Lu, K. Yamamoto, L.H. Nomura, S. Mizuno, Y.M. Lee, and R. Thawonmas, "Fighting Game Artificial Intelligence Competition Platform," Proc. of the 2013 IEEE 2nd Global Conference on Consumer Electronics, pp. 320–323, 2013.

[6] P. Rohlfshagen and S.M. Lucas, "Ms Pac-Man versus Ghost Team CEC 2011 Competition," Proc. of the 2011 IEEE Congress on Evolutionary Computation, pp. 70–77, 2011.