Learning to Play Fighting Game using Massive Play Data

Hyunsoo Park, and Kyung-Joong Kim*
Department of Computer Science and Engineering
Sejong University, Seoul, South Korea
hspark@sju.ac.kr, kimkj@sejong.ac.kr

Abstract—Designing fighting game AI has been a challenging problem because the program should react in real-time and require expert knowledge on the combination of actions. In fact, most of entries in 2013 fighting game AI competition were based on expert rules. In this paper, we propose an automatic policy learning method for the fighting game AI bot. In the training stage, the AI continuously plays fighting games against 12 bots (10 from 2013 competition entries and 2 examples) and stores massive play data (about 10 GB). UCB1 is used to collect the data actively. In the testing stage, the agent searches for the similar situations from the logs and selects skills with the highest rewards. In this way, it is possible to construct the fighting game AI with minimum expert knowledge. Experimental results show that the learned agent can defeat two example bots and show comparable performance against the winner of 2013 competition.

Keywords—Fighting game AI; Multi-armed bandits problem; UCB1; Game AI competition

I. INTRODUCTION

The fighting game has been one of the favorite video game genres and usually played by two human players. Although computer programs have been used to enjoy human players, they're designed manually by experts. Recently, the fighting game AI competition is introduced to promote the invention of new AI techniques for the game. In 2013 competition, the most of entries were designed manually. However, Mizuno AI from the organizer introduced the use of machine learning (K-Nearest Neighbor) after the last year's competition.

The design of game AI from the player's logs has been challenging research issue. For example, Cho *et al*. used human expert's replay files to build a strategy prediction model using machine learning [1]. It shows that the use of play logs can be useful on the design of game AI. The process includes logging of game playing from human players and/or bots. From the data, the computer program learns the mapping between game states and proper actions.

In this paper, we propose to build the fighting game AI from the matches against bots. The most significant problem is that it is impossible to collect data for all possible states and actions. Because the fighting game considers several factors to select actions, it is not a trivial task to test all possible game states and actions combinations. It is necessary to collect data efficiently to maximize the chance of high rewards.

The identification of good actions for the given circumstance can be defined as multi-armed bandit problem [2] which suffers from the exploration-exploitation dilemma. For instance, if there are N slot machines, a player can choose to pull the lever of machines M times in total and each slot machine's expectation reward is different. How the player can maximize the reward? If we know the "true" reward expectations of each machine, we can pull the lever of the highest reward machine (exploitation). Unfortunately, we don't know the "true" reward expectation of each machine without many trials. Therefore, we need experiments to get estimation of reward (exploration). If the given chance M is limited, then we have to balance between exploitation and exploration.

We propose to collect game data actively focusing on the cases with high rewards. During the data collection stage, our bots selects an action against opponents using UCB1 (upper confidence bound 1). In order to test our idea, we use the fighting game AI competition platform [3]. This platform supports basic fighting game components and bot API. Anyone can easily implement own entry and submit it to the competition. In this work, we consider the first order Markov decision process which simplify the inference of the actions from the collected data.

II. PROPOSED METHOD

We define all conditions around the AI player as state, and there are a list of possible actions at each state (11 skills + 1 key combination for guard). Reward is defined as the difference between "hit" to the opponent and "damage" from him after short time delay. The reward is the summation of difference of between hit to opponent damage (h) and from opponent (d) for 120 frames after the action execution (reward=$\sum r^i(h_i-d_i)$, i: frame index from action start, r: degrading factor, 0.98).

Our goal is to find good policies (state-action mapping with high reward) for the current state. In the training stage, our bot collects game logs (state, action, and reward). In addition to the sensory data from the fighting game, the player adopts "K-Nearest Neighbor" during the match to predict the next actions of the opponent. The estimation is also stored in the "state."

UCB1 is one of popular algorithms to handle the exploitation and exploration dilemma [2]. This algorithm

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (2013 R1A2A2A01016589, 2010-0018950).

*: corresponding author

978-1-4799-3547-5/14/1.00

selects the action j that gets the highest UCB1 value (equation 1). It linearly combines estimated expectation reward ($\bar{x}_j$) and curiosity (right term) of specific action j (n: # of trial of all actions, n_j: # of the action j's trial). Eventually, this value becomes higher when the action j's reward is high and/or curiosity term is high. The UCB1 is used to collect the data actively.

$$UCB1 = \bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}} \quad (1)$$

A hashing function is introduced to speed up searching for the most similar situation. The input to the function is the distance between players, skills of opponent, and estimated skills. The hashing function is used to quantize the huge number of games states into the buckets. In the actual match, our agent identifies the bucket using the current game state and selects an action with the highest reward. Fig. 1 summarizes the data collection and playing games with the data.

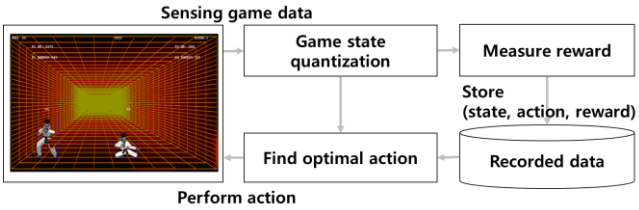

Fig. 1. Overview of the proposed method

Fig. 2 shows the operation of action selection for our learned agent. The hash function returns a key for the current game state. For each key, there is a list of game logs storing state-action-reward sets. If the number of data is N, there are N pairs of action and rewards. For each action, the agent averages all rewards for the action. It selects an action with the highest average of rewards.

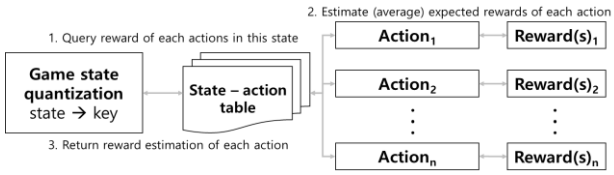

Fig. 2. Reward estimation method

III. Experimental Results

In the training stage, our program plays fighting games against 12 bots (ten 2013 entries and two examples (random AI and Mizuno AI). The random AI presses the key's on/off randomly. The Mizuno AI uses K-Nearest Neighbor algorithm. For each opponent, the agent plays 50 games. In total, it plays 600 games (10 hours). The size of data is 10GB (the logs are stored in JSON format). In the training data, there are state/action/rewards of each moment but the opponent player's name is not stored.

In the test, we use three representative players (random AI, Mizuno AI and "T" from 2013 competition). To the best of our knowledge, the Mizuno AI is the only agent with learning capability. And most of entries of 2013 competition was designed manually. "T" is the winning entry of the 2013 competition and is designed with finite state machine. In the last year's competition, ten entries was submitted. It showed that the "T" outperforms all other entries. However, we found that the Mizuno AI outperforms the "T." (Mizuno AI was not an entry of 2013 competition and included recently.)

In the game, the two players have three matches (about 3 minutes). The final scores are calculated based on the player and opponent's health power. We follow the scoring metric used in the 2013 fighting game AI competition. The sum of two players' scores is 1000. Table I summarizes the scores of the matches between the learned agent and the three players. It shows that our agent outperforms the "random AI" and the Mizuno AI. Although "T" is strong against our program, the scores acquired (387) is comparable to the scores (average 253) in the last year's entries against "T."

TABLE I. Performance of Learned Bots

	Opponent Bots		
	Random AI (example)	***T (1st rank)***	***mizunoAI (example)***
Learning + Simple Rules for Defense	**793.9** / 206.5	**387.3** / 612.6	**645.6** / 354.4
Learning	**799.8** / 200.1	**340.9** / 659.0	**595.4** / 404.5

* Average of 30 experiments, trained bot's point / opponent bot's point

IV. Conclusions and Future Works

In this work, we propose to use data-driven learning for the design of fighting game AI competition. The agent uses massive play data to select actions given the situation. It can significantly reduce the effort to design the AI agent from trial and errors. In the data collection, UCB1 is used to collect data effectively with a balance of exploration and exploitation. Because the data is massive, hashing is used to accelerate the search speed when the agent plays an actual game. Experimental results show that the learned agent can be better than random AI and other learning-based controller (Mizuno AI). Also, our bots show comparable performance against the winner of 2013 competition.

As a future work, it needs to design a better hash function assigning similar number of samples into the buckets. Also, the current approach considers only immediate rewards from the actions. However, it is desirable to consider delayed rewards from a sequence of actions (combination of skills).

References

[1] H.-C. Cho, K.-J. Kim, and S.-B. Cho, "Replay-based strategy prediction and build order adaptation for StarCraft AI bots," in *2013 IEEE Conf. on Computational Intelligence in Games (CIG)*, 2013, pp. 1–7.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Mach. Learn.*, vol. 47, no. 2–3, pp. 235–256, May 2002.

[3] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas, "Fighting game artificial intelligence competition platform," in *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on*, 2013, pp. 320–323.