Application of Monte-Carlo Tree Search in a Fighting Game AI

Shubu Yoshida*, Makoto Ishihara†, Taichi Miyazaki*, Yuto Nakagawa†, Tomohiro Harada*, and Ruck Thawonmas*
Intelligent Computer Entertainment Laboratory
*College of Information Science and Engineering, Ritsumeikan University, Shiga, Japan
†Graduate School of Information Science and Engineering, Ritsumeikan University, Shiga, Japan
Email: ruck@is.ritsumei.ac.jp

***Abstract*—This paper describes an application of Monte-Carlo Tree Search (MCTS) in a fighting game AI. MCTS is a best-first search technique that uses stochastic simulations. In this paper, we evaluate its effectiveness on FightingICE, a game AI competition platform at Computational Intelligence and Games Conferences. Our results confirm that MCTS is an effective search for controlling a game AI in the aforementioned platform.**

I. Introduction

The authors' laboratory has been annually organizing the Fighting Game AI Competition at Computational Intelligence and Games (CIG[1]) Conferences since 2013. In this competition, FightingICE [1] is used as the game platform, and participants compete for superiority of their AI's performance. Till now, high-ranking entries are mainly rule-based AIs, which always conduct a same action in a same situation. Such AIs are easily beaten by other AIs that play on their weaknesses.

In order to solve this problem, we apply Monte-Carlo Tree Search (MCTS) to a fighting game AI. This kind of AI decides its next action by stochastic simulations. In the past, an MCTS based approach produced significant results not only in a board, turn-based game like Go [2], but also in a real-time game like Ms. Pac-Man [3]. However, unlike Ms. Pac-Man whose response time is 40 ms, FightingICE's response time is 16.67 ms, making it a more challenging application of MCTS.

In this paper, we evaluate the effectiveness of MCTS on FightingICE. For this, we implement an AI that is controlled by MCTS using Upper Confidence Bounds 1 ($UCB1$) as the tree policy [4]. This AI is tested against each of the top-five AIs in the 2015 Fighting Game AI Completion.

II. Monte-Carlo Tree Search AI

MCTS is a best-first search technique that uses stochastic simulations. Its basis is a game simulation where both AI and its opponent play random moves or better pseudo-random moves. MCTS builds and uses a game tree, according to the four steps described later. These steps are repeated a specific number of times or during allowed time budget. After this, the game action corresponding to the root node's child node that was explored (visited) most is conducted in the actual game.

[1]http://www.ieee-cig.org

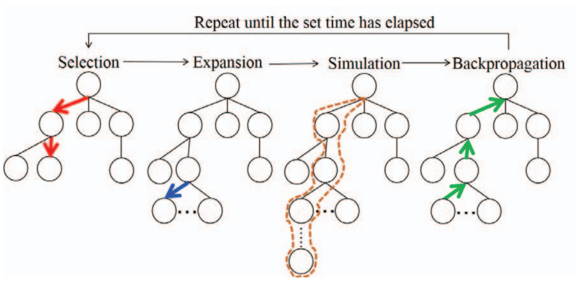

Fig. 1. An overview of MCTS for the proposed AI

Fig. 1 shows an overview of MCTS for our AI. In this figure, the root node represents the current game state while each of the other nodes represents an action. A path from the root node to a leaf node indicates the corresponding sequence of actions that will be first performed by the AI in a given simulation of T_{sim} frames, after which if there is room for further actions the AI will perform random actions for the remaining frames.

After expanding all adjacent child nodes to the root node, the following process is entered.

- **STEP 1 Selection**

A next action is chosen based on a given tree policy. This step is repeated several times until a terminal node is reached. As the tree policy, we use $UCB1$ (1):

$$UCB1_i = \overline{X}_i + C\sqrt{\frac{2\ln N_i^p}{N_i}}, \tag{1}$$

where C is a balance parameter, N_i is the number of visits at the i-th node and N_i^p is the number of visits at the parent node of the i-th node. In addition, $\overline{X}_i$ is the average reward (2):

$$\overline{X}_i = \frac{1}{N_i}\sum_{j=1}^{N_i} eval_j, \tag{2}$$

where $eval_j$ is the value of the reward at the j-th simulation given by (3):

$$eval_j = \left(afterHP_j^{my} - beforeHP_j^{my}\right) - \left(afterHP_j^{opp} - beforeHP_j^{opp}\right), \quad (3)$$

where the first term is the own hit-point (HP) difference before and after the j-th simulation while the second term is the opponent's one.

- **STEP 2 Expansion**

If the search reaches a terminal node whose number of visits is larger than the threshold N_{max} and the depth of the game tree is smaller than the threshold D_{max}, then all adjacent child nodes are added.

- **STEP 3 Simulation**

A simulation is performed for T_{sim} frames. As stated earlier, the AI will perform a sequence of actions in the path from the root node to the current leaf node, and after this it will perform random actions until the simulation-time budget of T_{sim} frames is reached; concurrently, the opponent will perform random actions. Finally, a reward is calculated according to (3).

- **STEP 4 Backpropagation**

After reaching the end of simulation, an update is performed for the $UCB1$ value of each tree node that was traversed in the path.

In this paper, the above four steps are repeated until T_{max} is reached.

III. Experiment

A. FightingICE Game

A game consists of three rounds, each having 60 s. The winner of a match is the AI whose sum of the scores in each round, $Score_{my}$, given below, is the larger.

$$Score_{my} = \frac{HP_{opp}}{HP_{my} + HP_{opp}} * 1000, \quad (4)$$

where HP_x is the amount of HPs the AI (x = "my") or the opponent (x = "opp") has received since the beginning of the corresponding round. In this paper, we conducted an experiment using FightingICE Version 1.23, the official version of the 2015 competition.

B. Experimental Method

The proposed AI played games against each of the top-five AIs in the 2015 Fighting Game AI Competition. All of them are rule-based AIs. For each opponent, we conducted 100 games by switching the initial position (left or right) of each AI every 50 games. We then compared the average scores the proposed AI obtained against each opponent AI. The aforementioned parameters in the proposed AI were empirically decided as follows: $C = 3, N_{max} = 10, D_{max} = 2, T_{sim} = 60, T_{max} = 16.5$s.

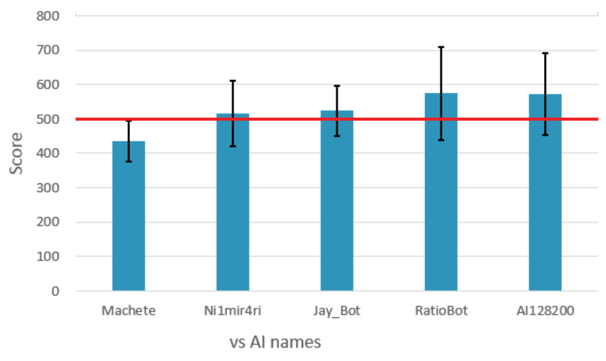

Fig. 2. Average scores against each AI

IV. Results and Discussions

Fig. 2 shows the average scores that the proposed MCTS-based AI obtained against each opponent AI in 100 games. In this figure, the vertical axis represents the average score while the horizontal axis represents the opponent AI in descending order of the 2015 competition ranking. As can be seen, the proposed AI outperformed (with score more than 500) all opponent AIs, except for the 1st ranked AI (Machete).

As for the performance against Machete, Machete is a well tuned rule-based AI that repeatedly conducts short actions, requiring less number of frames, which prevents actions by the proposed AI from being effective. This kind of opponent behavior is not considered in MCTS simulations. As a result, the proposed AI cannot effectively hit its own actions against Machete.

V. Conclusions

In this paper, we applied MCTS to a fighting game AI. Our results show that MCTS is an effective mechanism for controlling a game AI in a fighting game, which is a real-time game with a short response time. In the simulation step, the proposed AI simulates the opponent's actions at random. In the future, we plan to add an opponent modeling mechanism, like the one adopted in our previous work [5], and exploit it to better simulate the opponent.

References

[1] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas. *Fighting Game Artificial Intelligence Competition Platform*, Proc. of IEEE 2nd Global Conference on Consumer Electronics (GCCE), pp.320-323, 2013.

[2] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvari, and O. Teytaud. *The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions*, Communications of the ACM, Vol. 55, No. 3, pp. 106-113, 2012.

[3] K. Q. Nguyen and R. Thawonmas. *Monte-Carlo Tree Search for Collaboration Control of Ghosts in Ms. Pac-Man*, IEEE Transactions on Computational Intelligence and AI in Games, Vol.5, No. 1, pp. 57-68, 2013.

[4] L. Kocsis and C. Szepesvari. *Bandit Based Monte-Carlo Planning*. Proc. of the Seventeenth European Conference on Machine Learning, pp. 282-293, 2006.

[5] K. Yamamoto, S. Mizuno, C.-Y. Chu, and R. Thawonmas. *Deduction of Fighting-Game Countermeasures Using the k-Nearest Neighbor Algorithm and a Game Simulator*, Proc. of 2014 IEEE Conference on Computational Intelligence and Games, pp. 437-441, 2014.