MONTE-CARLO PLANNING FOR UNIT CONTROL IN STARCRAFT

Zhe Wang, Kien Quang Nguyen, Ruck Thawonmas, and Frank Rinaldo
Intelligent Computer Entertainment Laboratory
Ritsumeikan University

ABSTRACT

This paper presents an application of Monte-Carlo planning (MCPlan) to controlling units in a RTS game StarCraft. We apply an $\varepsilon - greedy$ algorithm to model the opponent in a simulation for improving MCPlan's performance. Experimental results are provided at the end of this paper, which show the potential of MCPlan in this domain.

I. INTRODUCTION

StarCraft is one of the most popular RTS game developed by Blizzard Entertainment. The extremely balanced gameplay and easy access to the game engine not only provides players with multiple options, but also enables it to be an ideal platform to test various AI approaches. However, because of its real-time property and largely unpredictable randomness, it is a challenging game for AI research.

In this paper we focus on the micro management of StarCraft gameplay. Micro-management is a series of actions that issue commands to each unit of a certain group for maximizing their effectiveness during combat. It is a significant part in RTS game because it can bring players great advantages in the game.

Monte Carlo Planning

MCPlan is a mechanism based on simulation and does a stochastic sample of possible choices. It is an effective method to handle random and imperfect information issues with alternating moves by determining the best result after multiple roll-outs. A great advantage of MCPlan is the reduction of expert knowledge required, it only need an effective evaluation function.

Previous work has been done on both unit control and the MC-simulation application. Unit control is usually handled by finite state machines, scripts or neural networks, etc. For example, Weber and Michael used hand authoring ABL behaviors to handle micro-management task in StarCraft [1]. A Bayesian model was applied to StarCraft for unit control by Gabriel and Pierre [2]. But they all need lots of expert knowledge. Monte-Carlo method in RTS games can also be found from previous work. Such example include works applying MCPlan to high-level planning [3], or combining Monte-Carlo method with non-linear value function approximation (VFA) and text recognition technique as a solution for large sequential decision making problems [4]. However, these works mainly focus on high-level planning, and MC simulation can rarely be found in low-level AI modules such as micro management.

The contributions of this paper are as follows:

- Implementation of the MCPlan with expert knowledge for micro-management in StarCraft.
- Design and description of a simulator for complex commercial RTS game combat scenario.

II. APPLICATION TO STARCRAFT

We apply MCPlan to Starcraft by accessing the game engine through BWAPI[1].

Simulator Design

The simulator is a model that is used to simulate the game for making predictions. In our case, the simulator creates possible game scenarios in the near future.

Our simulator includes three major modules:

A) *Character modeling* - Characters should be exactly the same as in the real game, including character's hit-points, attack range, special skills etc.

B) *Map modeling* - Map modeling should consider different terrains, characters' location, and unit overlapping issues.

*C) Opponent behavior m*odeling - We adopt an ε-greedy algorithm to define opponent movement in simulator. The evaluation function and plans for opponent are basically the same as MCPlan AI.

[1]For BWAPI please visit http://code.google.com/p/bwapi

978-1-4673-1501-2/12/$31.00 ©2012 IEEE

Simulation roll-out

Each roll-out contains two plans: one is fixed and the other one is stochastic. The fixed one is a specific plan chosen for evaluation, so it is always simulated first. After that, the program will randomly pick another plan and continue the simulation process. After simulating both plans, this roll-out is evaluated.

Search Algorithm

Here is our algorithm, UCB1, for searching the best plan.

$$UCB1(i) = R_i + C\sqrt{\frac{\ln N}{N_i}} \qquad (1)$$

In the formula, i is the index of each plan. R_i presents the reward that plan i obtains from the evaluation function. C is a predefined constant, N_i and N are the total procedure number of roll-outs and the number of times that plan i has been visited as the fixed plan respectively.

The basic view of our MC simulation is as follows:

1) Load plans one by one to the simulator.
2) Simulate each roll-out once, evaluate it and reward its fixed plan.
3) Choose the best plan that evaluation by UCB1, simulate the roll-out and reward the plan again.
4) Repeat step 3.
5) Choose the plan with best average result for the AI player in a real game.

Evaluation function

An evaluation function is for measuring the effectiveness of a roll-out plan in different situations. Our evaluation function is for individual units and contains 4 aspects:

$$Q = \omega_1 HP + \omega_2 DM + \omega_3 SP + \omega_4 EG \quad (2)$$

Four elements in this function are individual unit's hit-point, damage to opponent, move speed and remaining energy, respectively. We manually set their weights ω based on expert bias.

III. EXPERIMENTS AND RESULTS

We designed two experiments to test our MCPlan AI, both are combat scenarios of Terran against Zerg. In order to ensure all units get involved in the combat quickly, combats are limited to 16×16 (game size) space. The difficulty of two experiments is basically the same, but the environment in experiment 2 is more complex and requires different skills.

For highlighting the effectiveness of micro-control, Zerg force units always have military advantages and are controlled by the original AI. We then applied different subjects (original AI, MCPlan AI and human expert) to control Terran force and compare their performance. The results are as follows (Table I):

	Experiment 1	Experiment 2
	8 Marines vs 12 Zerglings	8 Marines vs 10 Zerglings and 1 Lurker
OAI	0%	0%
MCAI	33.3%	60%
Expert	80%	93%

Table I. Results of two experiments

The table above shows the win rate of all subjects, each experiment runs 15 times for each subject. We can see Monte-Carlo AI performs much better than Original AI in both experiments, but still incomparable to human expert.

IV. CONCLUSION AND FUTURE WORK

This paper presented a preliminary work for solving the problem of unit-control in RTS games. We described a mechanism for applying MCPlan to Starcraft. We successfully developed a simulator for the game and tested our method. Our work includes the design of plans, simulator, and evaluation function as well as their implementation. From the experiments, we obtained initial results which indicated a promising potential of MCPlan in this domain. For future work, we intend to improve the simulator speed and find a way to automatically decide the weights in evaluation function.

REFERENCES

[1] B. Weber, M. Mateas and A. Jhala, "BuildingHuman-Level AI for Real-Time Strategy Games", *AAAI Fall Symposium on Advances in Cognitive Systems (ACS 2011),* pp. 329-336.

[2] G. Synnaeve, and P. Bessiere, "A Bayesian model for RTS units control applied to StarCraft", in *IEEE Conference on Computational Intelligence and Games(CIG)*, 2011, pp. 190-196.

[3] M. Chung, M. Buro, and J. Schaeffer, "Monte carlo planning in RTS games," in *IEEE Symposium on Computational Intelligence and Games (CIG)*, 2005.

[4] S.R.K Branavan, David Silver, Regina Barzilay, "Non-Linear Monte-Carlo Search in Civilization II",*in Proceedings of IJCAI*, 2011, pp. 2404-2410.