Artificial Intelligence for Games

3ºProject

TÉCNICO LISBOA

Grupo nº4 e 10:

Pedro Borges, nº73883

Ricardo Martins, nº 76449

Ricardo Silva, nº 78414

Abstract

In this work, we implemented different variants on Monte Carlo Tree Search, with the aim to compare the variants on efficiency and effectiveness levels. The results obtained are presented below.

Test Section

To obtain this results we made 10 runs of the 4 implemented variants: MCTS, MCTSRAVE, MCTSBiased, MCTSBiasedRAVE. All of them were ran on the same machine with a i5-6200U, 2.8Ghz and 8gb ram. We noticed that with computers with different specifications the result varies slightly. Another thing to take in consideration it is the fact this results are hard to replicate because of the decision which each variant makes differ.

Efficiency Data

MCTS

Method	Calls (#)	Execution Time (ms)
MCTS.Run	1	40.57
MCTS.Playout	25	37.25
MCTS.Selection	25	2.98
MCTS.Expand	4	0.63
MCTS.Backpropagate	25	0.16
MCTS.BestUCTChild	63	1.38
MCTS.BestChild	11	0.11

MCTSRAVE

Method	Calls (#)	Execution Time (ms)
MCTS.Run	1	50.43
MCTSRAVE.Playout	25	39.96
MCTS.Selection	25	6.19
MCTS.Expand	6	0.93
MCTSRAVE.Backpropagate	25	4.08
MCTSRAVE.BestUCTChild	58	4.11
MCTS.BestChild	9	0.1

MCTSBIASED

Method	Calls (#)	Execution Time (ms)
MCTS.Run	1	119.69
MCTSBiasedPlayout.Playout	25	111.70
MCTSBiasedPlayout.Selection	25	7.49
MCTS.Expand	24	2.93
MCTS.Backpropagate	25	0.28
MCTS.BestUCTChild	104	2.63
MCTS.BestChild	13	0.12

MCTSBIASEDRAVE

Method	Calls (#)	Execution Time (ms)
MCTS.Run	1	98.83
MCTSBiasedRAVE.Playout	25	77.72
MCTSBiasedRAVE.Selection	25	10.00
MCTS.Expand	18	2.46
MCTSBiasedRAVE.Backpropagate	25	10.85
MCTSBiasedRAVE.BestUCTChild	96	5.61
MCTS.BestChild	11	0.15

Efficacy Data

MCTS	Number of Wins: 6 Number of Losses: 4 Win Rate: 60% Best Time: 162.5 Worst Time: 200 Worst Win Time: 180.5
MCTSRAVE	Number of Wins: 8 Number of Losses: 2 Win Rate: 80% Best Time: 154 Worst Time: 200 Worst Win Time: 183
MCTSBiased	Number of Wins: 10 Number of Losses: 0 Win Rate: 100% Best Time: 89 Worst Time: 101 Worst Win Time: 101
MCTSBiased RAVE	Number of Wins: 10 Number of Losses: 0 Win Rate: 100% Best Time: 82 Worst Time: 88 Worst Win Time: 88

Discussion

In terms of efficiency the vanilla variant of MCTS obtained the best results because is the least complex in terms of calculations and decision making.

Followed by this result is MCTSRAVE, though it adds an extra step in BestUCTChild by calculating 3 different values RAVE, MCTS and UCT choosing the best of the three and at MCTSRAVE.Backpropagate and extra loop is required to update the RAVE values on the child Nodes.

MCTSBiased and MCTSBiasedRAVE have the lowest performance, since both use more information of the world State to bias their decisions. Both include a heuristic calculation per executable action to influence the next random action. But at this cost we gained more efficacy.

At the efficacy level MCTSBiasedRAVE achieved the best results followed closely by MCTSBiased by having 100%-win rate and 7 seconds' difference on best times. Though MCTSRAVE obtained an 80%-win rate this is because he reached the time limit of the level not because of bad decision that gets him killed, this cannot be said for MCTS which reached an 60%-win rate by either reaching the time limit or making bad decisions.

Optimizations

Efficiency Level

In MCTSRAVE we reduced the number of time that the MCTS and Exploration values are calculated and reducing the number of time β and (1-β) values are calculated.

$$MCTSValue = \frac{ChildNode[i].Q}{ChildNode[i].N}$$

$$ExplorationFactor = C * \sqrt{\frac{\log(node.N)}{ChildNodes[i].N}}$$

$$RAVEValue = (1-\beta)MCTSValue + \left(\beta * \frac{ChildNodes[i].QRAVE}{ChildNodes[i].NRAVE}\right) + ExplorationFactor$$

$$UCTValue = MCTSValue + ExplorationFactor$$

Efficacy Level

In MCTSBiased the method Selection has two different variants.

The first variation chooses the next action by selecting the next non-repeated action with the lowest heuristic value making the MCTS tree starting by looking at the best decisions.

The second variation chooses the next action with lower heuristic value than the last made decision, forcing the search to incrementally make better decisions.

To achieve better results with the vanilla version of the MCTS and ordering of the actions by lower at the start-up of the creation of the decision-making algorithm.

Comparison of recursive implementation of the WorldModel and the FEAR style implementation.

This test was done using the MCTSBiasedRAVE algorithm. Due to the random nature of the algorithm we calculated the average time per call instead of just calls and total execution time because there was no way to subject them to the same conditions.

Recursive implementation:

Method	Calls	Execution time (ms)	Average Time per Call (ms)
getNextBiasRandomAction	175	35.80	0.2046
GetProperty	1303	4.13	0.003169
SetProperty	317	0.40	0.001262
GetNextAction	111	0.56	0.005045
GetExecutableActions	171	27.33	0.1598
IsTerminal	311	2.34	0.007524
GetScore	25	0.06	0.024
CalculateNextPlayer	199	5.58	0.02804

FEAR style implementation:

Method	Calls	Execution time (ms)	Average Time per Call (ms)
getNextBiasRandomAction	164	26.81	0.1635
GetProperty	209	0.11	0.0005263
SetProperty	123	0.59	0.004797
GetNextAction	132	0.36	0.02727
GetExecutableActions	156	19.25	0.1234
IsTerminal	141	0.52	0.03688
GetScore	25	0.03	0.0012
CalculateNextPlayer	116	2.54	0.02190

As we can see the second implementation is slightly faster however it is hard to compare them because the time per call varies a lot in different stages of the game and on different world states. We decide to use the new representation of world model anyways.