An Investigation of Using Monte-Carlo Tree Search for Creating the Intelligent Elements of Rise of Mitra

Thales Aguiar de Lima[1]* Charles Andrye Galvao Madeira[2]

[1]Universidade Federal do Rio Grande do Norte, Departamento de Informática e Matemática Aplicada, Brazil
[2]Universidade Federal do Rio Grande do Norte, Instituto Metrópole Digital, Brazil

Rise of Mitra

Figure 1: Rise of Mitra logo.

ABSTRACT

Games have been used as a good test environment for AI. This paper describes the outcomes of Monte-Carlo Tree Search (MCTS) in the AI of Rise of Mitra(RoM), a discrete world turn-based game. This algorithm consists of Selection, Simulation, Expansion and Backpropagation phases. MCTS had successful outcomes in two-player games with perfect information like Go, where it managed to win some competitions. Therefore, the main goal of this work is to reach a challenging AI that can contribute to creating a more realistic and immersive game, besides being capable to consider uncertainty elements. Some strategies like OMC and UCB were used. Both of them resulting in a good win rate, approximately 70% for OMC and 60% for UCT. Therefore, this algorithm can win against normal players, but not at a point where a player can get frustrated, in other words, it allows the player to be challenged and more immersed in the game world.

Keywords: Monte-Carlo Tree Search, Game AI, Uncertainty, Rise of Mitra.

1 INTRODUCTION

Gameplay is a process highly affected by the player's interactions with the virtual world. Gameplay experience, as said in [10] is a complex process in which several player characteristics are combined with a meaning-making gameplay setting. Immersion addresses how much a player becomes distracted from the surrounding real world and creates empathy to the simulated game world. In current games, this is a carefully treated topic since that means how much time players will spend playing their games, or also how they will feel about the game.

Being capable of creating elements with persuasive behavior may contribute to a more realistic player interaction with the game world. Most current games have poor, not challenging, predictable AI which takes totally random actions. These approaches are likely to create poor feedback and prejudice the player experience as shown in the model [17]. Thus, creating an AI with a decision-making process that takes into consideration the player's behavior and also learns from it can create a more immersive world. The game Alien: Isolation shows a great example of how a good AI can contribute to that aspect[1] where there is an enemy that can change his behavior, spending more time nearby the player or improving his search by looking at places he already found the player hiding.

Therefore, this paper describes a Monte-Carlo Tree Search (MCTS) AI-based for the game Rise of Mitra(RoM). It is a new algorithm family that has reached good outcomes in games with perfect information, like Go or Chess. MCTS is an incomplete tree search algorithm that uses several kinds of statistics to choose a valid move.

This paper is organized as follows: section 2 describes Rise of Mitra story and mechanics. Following that, section 3 is a description of MCTS Algorithm and a brief state of the art. Next is where the algorithm functions, weights, and decision-making are shown, in other words, that section explains how domain knowledge is inserted into the algorithm. Finally, the results are shown and followed by conclusions.

2 THE GAME: RISE OF MITRA

Rise of Mitra is being developed by the author. Its source code can be downloaded in [2]. This section will introduce the game story and mechanics to help understand the decisions taken in the algorithm.

2.1 Story

Rise of Mitra tells the history of a small planet called Mitra, where an ancient battle for resources between two races takes place. This planet is the only place in the galaxy where Argyros Crystal grows. A small piece of it has an almost infinite energy source that can be used to sustain a large city for many centuries. Rakhars and Dalrions inhabit the planet. The Rakhars are a technological race. They are half organic and mechanical species that have a knowledge superior to that the Dalrions. Whereas the Dalrions are a more archaic culture, venerating the old gods and making sacrifices in their names. Both of them have their interests on Argyros, and they have been battling for the control of this resource since ancient times.

The following section describes what is a valid move in the game, how they work, illustrates an idea of the game board, and describes all the units of Rise of Mitra.

2.2 Mechanics

RoM is a two-player turn-based board game composed of a discrete world, this means that the characters can move only in a determined cell with constant size, for instance, common games of this genre are Go, Chess and Risk. In turn-based games, each player takes turns when playing and only one movement can be performed per turn. Moreover, RoM also has imperfect information, added by a decision tree. The main objective of RoM is to destroy the opponents culture representation, called *Cultural Center*. Rise of Mitra

*e-mail: thalesaguiar21@gmail.com

Figure 2: RoM board example. The green squares are the moving range, while red squares are attack range.

has the following units: *Cultural Center* and *Pawns*. The former represent the race culture and can generate more Pawns. If this unit gets destroyed, its owner loses the game. The latter is the movement unit, it can move around the board, attack, and defend.

The board consists of a 25×35 grid separated into sections. Each section, called *Terrain* is a set of cells with nature in common, modifying the attributes of game units while they are in this terrain. Table 1 shows how each pawn is affected by the Terrains and Figure 2 shows an example of RoM board where blue icons are Dalrion's units and yellows are Rahkars, dots represent an empty cell and uppercase x are blocked cells. The agglomerated '@' and '%' are, respectively, the Dalrion and Rahkar cultural center.

Table 1: Terrain effects on pawns

Terrain	Dalrion	Rakhars
Mountain	-1 MOV	+1 MOV
Plain	+1 ATK	-1 DEF
River	+1 ATK	-1 MOV
Field	+1 DEF	+1 DEF
Marsh	-1 DEF	+2 ATK
Forest	+1 MOV	+1 ATK
Desert	+2 MOV	-1 ATK

Each player starts with 6 pawns. Each pawn has health, attack, defense and movement points. They can move, attack enemies, and defend enemies attacks. They die if their health is less than or equal to zero. A player can not have more than 6 pawns, and the Cultural Center will create more pawns in defined turns. Valid movements in RoM are **Move** a pawn n tiles, where $n \leq movepoints$ using Manhattan distance, **Attack** an enemy unit within k tile distant, where k is the pawn's attack range, and **Defend** an enemy attack. In case the enemy's attack points are less than or equal to the allied defense, then the ally takes no damage.

Unit's attributes were pseudo-randomly selected from two vectors consisting of 5 values which sum 27 points each. Moreover, Dalrions have 1 movement and 5 extra life points, whereas Rahkar has 2 defense and 1 attack range.

Now, with a basic understanding about RoM, next section gives an introduction for the MCTS algorithm which is the main goal of this paper.

3 Monte-Carlo Tree Search

Monte-Carlo Tree Search(MCTS) is an adaptive incomplete tree search method that uses statistical values in the decision-making process. Another definition is a procedure for finding optimal decisions, based on probabilistic values, for a problem by extracting

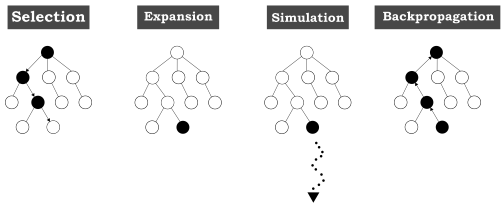

Figure 3: Monte-Carlo phases illustration.

random samples in a given solution space and creating an incomplete search tree in memory[4]. This is a good definition because it elucidates all four phases of an MCTS program, each of them will be better described further on.

MCTS is a fairly new algorithm family, which has been successfully applied in games like Poker[11], Pac-man[16], Go[13][12][4] and Settlers Of Catan[18]. Each of these games has different genres, attributes, and gameplay. Therefore MCTS can be applied to different sorts of games, even with hidden information. In poker, it is not possible to see which cards the opponents are holding. Consequently, MCTS can be seen as an independent game algorithm, as defined by [7], and can be implemented as a framework for every game [5].

In [13] the authors were able to obtain a win rate between 50% and 60% using the rapid value estimation MC-RAVE against GnuGo 3.7.10 at level 10 in MoGo with a 9×9 grid. Also [4] obtained the best results with the simulation technique based on [6] and won several competitions. Furthermore, in *Settlers Of Catan*[18] the writers used MCTS to play against the game AI obtained approximately 50% win rate and an average score of 8.3 while [11] with their MCTS adapted to consider uncertainty were able to get an average profit of 150 against other specific bots like RuleBotBestHand, which consider the probability that he has the best hand and take his decisions based on that, and about 15 against standard bots.

MCTS has four phases: Selection, Expansion, Simulation, and Backpropagation. In *Selection* phase, the algorithm searches for the move to be played from the tree root to the leaves. Then, the *Expansion* will add nodes to the tree if the Selection reached a non-finishing leaf node. Next, *Simulation* phase will play k random games with pseudo-random valid actions using Monte Carlo simulations. Finally, the *Backpropagation* will update each node in Selection path with a given metric(win rate, for example). These behaviors were adapted from [7].

MCTS is mostly applied to games with perfect information, which is not the case RoM. In [5], the author evaluates every legal move of the game *Settlers of Catan* using his knowledge about the game to balance the moves according to what is worth playing. For instance, creating more settlements has a high weight because it can provide more resources which are used to buy more settlements, roads, trade with other players, etc.Therefore, the purpose of this article is to combine the successful outcomes from the ideas in [5] and the uncertainty handling used by [11].

4 Domain Knowledge

This section describes which game characteristics are used by the Monte-Carlo Tree Search algorithm(from now on described as MUSASHI) and the Decision Tree(from now on described as Gaia). Adding knowledge to the algorithm stages can effectively increase its play strength. In [7] it is possible to see that even heuristic functions may be added to common strategies for each stage.

In Rise of Mitra, all information is contained on the board. This means that MUSASHI will create a partial RoM game tree in the

memory, where each node is a different game state. A Game Tree, as defined by [8], is a tree where the root represents a game's initial state before any moves have been taken. Each level below it represents all the possible outcomes from a valid move in a game and leaf nodes are final states. Therefore, each edge is one possible move of the game, for instance, the ones described in section 2.2. Moreover, it is common for each level to represent a move of a different player, so in a two player game like RoM the first level is the outcome of a valid move from player one, the second level is the outcome of player two move, and so on. This work evaluated each node(move) by the given constraints:

1) Every unit at ally side of the board will have move and attack values increased if there are enemies close to the Cultural center. Moving in the enemy's direction or attacking them.

2) An ally pawn that has attack range to hit the cultural center will have this move weight increased by 15.

3) Attacking an enemy in the range has an increased weight of 10.

4) If there is an enemy at a maximum distance of 10 then moving an ally in its direction will have an increased weight of 10, rising as it gets close to the enemy.

5) Moving to a cell within a terrain that gives a positive bonus to the pawn has an increased weight of +1 otherwise -1.

6) Moving towards the enemy's Cultural Center has its weight increased the closer the pawn is to it.

Next sections will explain which strategy we used for each MCTS phase. The last section briefly describes Gaia's uncertainty.

4.1 Selection

The selection phase controls the balance between exploration and exploitation. Therefore, as said in [18], high or low restrictions will make MCTS weaker, by restricting or selecting a bad set of actions. Accordingly to [7], some strategies often used are PBBM(Probability to be better than best move)[9], UCT(Upper Confidence Bounds to Trees)[14] and OMC(Objective Monte Carlo)[6].

The OMC uses a Urgency function $U(i)$ to determine the immediacy of each node

$$U(i) = erfc\left(\frac{v_0 - v_i}{\sqrt{2}\sigma_i}\right) \tag{1}$$

Where $erfc$ is the complementary error function, v_0 is the best move value, v_i and σ_i are respectively the value and standard deviation of move i. Furthermore, a fairness function selects the move considering the visit count n_j of node j, given by the equation 2 where S_i is the set of i siblings.

$$f_i = \frac{n_i \cdot U(i)}{n_j \cdot \sum_{j \in S_i} U(j)} \tag{2}$$

Another strategy used in this work is the UCT[14] used in Mango, an adaptation of UCB[3] for trees. UCT is a confidence bound based on the visit count of the current node n_i and the direct ancestor n_p. This strategy selects the node that satisfies the equation $m \in argmax_{i \in I}\left(v_i + \beta \cdot \sqrt{\frac{\log n_i}{n_p}}\right)$, where v_i is the node i value and β is a coefficient that can be experimentally chosen[7].

In [7] the author defines some node types. The *Max Child* node has the highest value, a *Robust Child* has the highest visit count, the *Robust Max Child* has both highest value and visit count and the *Secure Child* maximizes a lower confidence bound given by a function l. Each of these can be used as the selected movement. This work used different kinds of nodes, but without significant change in results.

The following section shows the Expansion phase and its close relationship with the Monte-Carlo Simulations and the memory management.

4.2 Expansion

The expansion stage will add nodes to the game tree. Here, it may evaluate if adding a given node is worth. This stage is responsible for the memory management, and it is important because, for instance, determining the winner in an $n \times n$ Go board is a PSPACE-Hard problem[15]. This task can be done in many ways, but MUSASHI will add one node per simulated game[9]. This simple and efficient strategy was used without a big loss in playability, that is, not choosing bad moves very often. Another strategy is to add only the Robust Child. This has a better chance of being a good move since Monte Carlo simulations will tend to choose the best move more often. Next, this paper describes the simulation stage and how it is applied with the algorithm.

4.3 Simulation

In this stage, the Monte-Carlo simulations take place by choosing actions accordingly to a given strategy, which can be random. In [7] the Urgency-Based Simulation is named, in which given an urgency function U, the immediacy value is calculated for every possible move at the current tree level and the probability p_j for each move j is given by $p_j = U(i)(\sum_{k \in M} U_k)^{-1}$ (3) where M is the set of possible moves.

Besides, in this paper proposes to use a function which given a set of states it selects the ones that respect a specified range. In this case, we can use the following equation $Q = \{i \in S | \bar{S} - \sigma_S > v_i\}$ (4). There, S is the set of valid moves and v_i is the value of move i. With this equation, the AI may explore a greater number of nodes, thus allow it to have a higher exploration rate.

Furthermore, the proposed algorithm will have a specified max depth to control simulations exploitation and use a function to restrain the exploration, in this case, the simulation strategies named before. Next, the Backpropagation phase is described.

4.4 Backpropagation

At this stage, the algorithm will update the simulation outcome of every visited node in the current play out. For this, there are several strategies and some of them were described by [7]. Even though they can increase the Monte-Carlo Tree Search game play, the use of strategies in this stage did not have a significant outcome[7] when compared with applying strategies, or domain knowledge, to other stages. For its simplicity and clarity, this work uses the win rate (the number of wins divided by the number of games played) which is the most effective and popular. Therefore, given a node n every ancestor node visited will receive $+1$ if the leaf node was a win, -1 if it was a loss and 0 otherwise.

4.5 Uncertainty

Gaia is a Decision Tree that checks if the current turn is multiple of a random number from 10 up to 15 (these numbers can be given, so 10 and 15 should be seen as an example), the cultural center risk, the mean distance between allies and from enemies to it, and the number of allies at each type of terrain. The Cultural center risk is calculated by (5) where S is the set of enemies, D_i is the i-th enemy distance from the allied center and C_l is current center's life. With this information, Gaia randomly changes positively or negatively unit's attributes.

$$C_r = \exp\left(\frac{1}{C_l}\right) + \exp\left(\frac{1}{N_a}\right) + \sum_{i \in S}\left(\exp\left(\frac{1}{D_i}\right)\right) \tag{5}$$

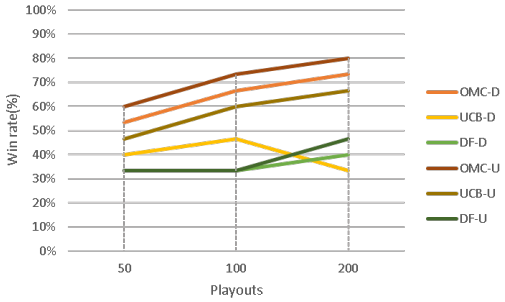

Figure 4: Win rate (vertical axis) of each selection strategy against player-03R with the number of play outs (horizontal axis).

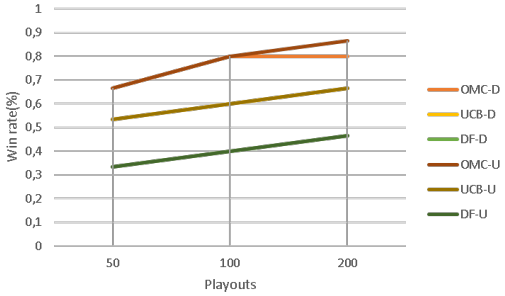

Figure 5: Win rate (vertical axis) of each selection strategy against player-05R for the number of play outs (horizontal axis).

Therefore, MUSASHI has to be able, in execution time, to store and classify moves that may trigger a good or bad change from the Decision Tree. For that MUSASHIstores each Game Tree node where Gaia was activated and tries to create a relation between the current move and the stored states.

5 RESULTS

This section presents the outcomes of the proposed algorithm against a random player-03R which randomly chooses a move that satisfies $v \geq (1-\beta)b$, where v is the value of the current node, b is the highest value, and β is a value between 0 and 1 (for 03R its 0.7). Therefore the player-05R take actions that satisfies the same equation with $\beta = 0.5$. Figure 4 shows the win rate of each technique against player-03R, while figure 5 has the result of playing against 05R. In both images -U means that equation 3 were used, and -D for the equation 4. Note that the outcomes are based on 15 games for each number of play out.

Furthermore, some tests with human players were made. The test was made with 4 subjects and each one played at least 10 games. After that, they were submitted to a quiz with questions about their game play experience, but mainly about the AI. They evaluated the AI difficulty and how they felt about it, where 75% of the subjects did not felt challenged by the AI.

6 CONCLUSION

Monte Carlo Tree Search algorithms allow virtual games to have a strong AI by exploring and exploiting the Game Tree. Besides, the proposed algorithm let it consider even uncertainty allowing it to become more applicable to modern games. The development of this AI was satisfactory to the author, by providing more knowledge about the fields state of the art. Furthermore, this paper presents a brief review of MCTS and thus is a good resource for initial studies.

The outcomes from the previous section showed that against the random players (player-03R and player-05R) the AI could reach good results. However, when playing against humans their review was that it is not challenging. Even so, they also said that it was not frustrating and the difficulty was not so high, that is a good outcome because it means players could have fun and also felt rewarded while playing. But, by not being characterized as a challenging AI the work did not fully reach its purpose and this can be addressed in future works.

REFERENCES

[1] Alien isolation's artificial intelligence was good...too good. Accessed: 4/26/16.

[2] Rise of mitra.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[4] B. Bouzy and B. Helmstetter. Monte-carlo go developments. *Advances In Computer Games*, 135:159–174, 2004.

[5] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game ai. In C. Darken and M. Mateas, editors, *Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 23–25. Association for the Advancement of Artificial Intelligence, October 2008.

[6] G. Chaslot, J. Saito, B. Bouzy, J. Uiterwijk, and H. V. den Herik. Monte-carlo strategies for computer go. In *Proceedings of The 18th BeNeLux Conference on Artificial Intelligence*, pages 83–90, 2006.

[7] G. M. J.-B. Chaslot. *Monte-Carlo Tree Search*. PhD thesis, Dutch Research School for Information and Knowledge Systems, September 2010.

[8] B. Coppin. *Artificial intelligence illuminated*. Jones & Bartlett Learning, 2004.

[9] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International Conference on Computers and Games*, pages 72–83. Springer, 2006.

[10] S. de Castell and jennifer Jenson. *Worlds in Play: International Perspectives on Digital Games Research*. Peter Lang Publishing, New York, Brodway, 2007.

[11] G. V. den Broeck, K. Driessens, and J. Ramon. Monte-carlo tree seach in poker using expected reward distributions. In J. S. R. Goebel and W. Wahlster, editors, *Advances in Machine Learning*, volume 1, pages 367–381, Nanjing, China, November 2009. Springer.

[12] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud. The grand challenge of computer go: Monte carlo tree search and extensions. *Magazine Communications of the ACM*, 55:106–113, March 2012.

[13] S. Gelly and D. Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175:1856 – 1875, July 2011.

[14] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[15] D. Lichtenstein and M. Sipser. Go is polynomial-space hard. *Journal of the ACM (JACM)*, 27(2):393–401, 1980.

[16] T. Pepels, M. H. M. Winands, and M. Lanctot. Real-time monte carlo tree search in ms pac-man. In G. Kendall, editor, *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAME*, volume 6 of *3*, pages 245 – 257. IEEE, September 2014.

[17] P. Sweetser and P. Wyeth. Gameflow: A model for evaluating player enjoyment in games. In *ACM Computers in Entertainment*, volume 3 of *3*, pages 3–27. Computers in Entertainment, July 2005.

[18] I. Szita, G. Chaslot, and P. Spronck. Monte-carlo tree search in settlers of catan. In *Advances in Computer Games*, pages 21–32. Springer, 2009.