UNIVERSITÀ DI PISA

Department of Computer Science

ARTIFICIAL INTELLIGENCE FUNDAMENTALS

FusionRTS

Project Report

Professor:

Vincenzo Lomonaco

Teaching assistant:

Elia Piccoli
Luigi Quarantiello

Students:

Enrico Calandrini (565430)
Carmine Vitiello (578070)
Davide Mele (690636)

Academic Year 2024/2025

1 Introduction

FusionRTS is a project developed as part of the Artificial Intelligence Fundamentals course (2024/2025) to explore advancements in AI for real-time strategy (RTS) games. Specifically, the project focuses on microRTS, a minimalist framework for RTS games that simplifies traditional mechanics while retaining essential strategic and tactical elements. This simplified environment allows for focused experimentation with AI systems.
The primary objective of FusionRTS is to address limitations in the existing Monte Carlo Tree Search (MCTS) implementation within microRTS. By integrating multiple known enhancements, FusionRTS aims to create a more robust and efficient AI capable of superior decision-making.

1.1 Why microRTS?

MicroRTS, developed in Java as open-source software, offers a deterministic and real-time environment for studying AI in RTS games. Unlike complex RTS games such as StarCraft, microRTS reduces complexity while maintaining critical strategic elements. This makes it an ideal platform for testing AI algorithms, particularly MCTS, a popular choice for decision-making in games.
MicroRTS includes several hard-coded AI strategies such as RandomAI, WorkerRush, LightRush, MiniMax, and MCTS. These pre-existing AIs provide benchmarks for evaluating new approaches.

1.2 Objectives of FusionRTS

As already mentioned, FusionRTS addresses the limitations of the existing NaiveMCTS implementation in microRTS and improves its performance. This project integrates several well-known enhancements to the MCTS algorithm, creating a more efficient and robust solution [5].
Moreover, each year the microRTS competition features a round-robin tournament that includes all newly submitted AIs alongside pre-existing ones used as benchmarks. The results from the past year competition can be found here. In our research, we aim to replicate this tournament by evaluating all AIs generated through different enhancement combinations, as well as the baseline AIs available in microRTS, to assess the impact of our improvements.

1.3 GitHub repository

All the created files and also a microRTS copy, can be found at this link `https://github.com/enri07/FusionRTSProject`. Inside of it, you can find a *README.md* file containing all the instructions needed to properly install and setup microRTS along with the newly created AIs.

2 MicroRTS AIs

In this section we provide an overview of the *NaiveMCTS* AI currently implemented in microRTS which should help to understand how the theoretical enhancements were incorporated. Then, we will move on to describe the implemented enhancements, which combinations will lead to different AIs.

2.1 NaiveMCTS

Monte-Carlo Tree Search (MCTS) [3]-[2] is a best-first search algorithm that incrementally constructs a search tree while using Monte-Carlo simulations to estimate the value of game states. The algorithm starts with only the root node and, until the computational budget is exhausted, it repeatedly performs simulations. Each simulation consists of four steps illustrated in Figure 1.

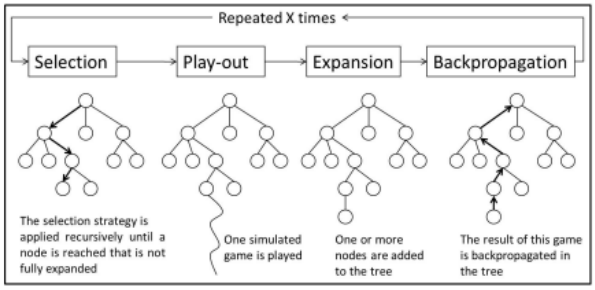


Figure 1: The four steps of an MCTS simulation. Adapted from [5].

A key component of the algorithm is the *selection policy* used during the *Selection phase*, which determines the best node to expand. In the NaiveMCTS implementation available in microRTS, two approaches are provided: a *Greedy* strategy and a UCB1-based method [3]-[1], where the successor S_i of the current node P is chosen by maximizing Equation 1. For further clarification regarding this formula, please refer to [5].

$$UCB1(S_i) = \overline{Q}(S_i) + C \times \sqrt{\frac{\ln(n_P)}{n_i}} \tag{1}$$

In our study, we decided to use NaiveMCTS as a starting point implementing the UCB1 strategy as it is the most commonly implemented selection policy.
Another critical aspect to emphasize is that, in the NaiveMCTS implementation, each node in the search tree considers moves for only one player. As a result, a single node does not fully represent a new game state, since transitioning to a new state requires actions from both players. To accurately capture a new state, it is necessary to examine nodes at depth 2, which account for the moves of both player 1 and player 2, rather than just nodes at depth 1. This detail becomes particularly significant when discussing the tree reuse enhancement, as explained later.

2.2 Progressive History

Progressive History [4] represents an enhancement to the Monte Carlo Tree Search algorithm by introducing a history of actions chosen during previous evaluations of the same game. Each action recorded in this history keeps track of the number of times it has been selected (n_a) and its corresponding evaluation (s_a). These values are integrated into the evaluation function described in 1, which determines the node to explore or develop during the search process. The new relation used for node selection is as follows:

$$PHUCB1(S_i) = \overline{Q}(S_i) + C \times \sqrt{\frac{\ln(n_p)}{n_i}} + \frac{s_a}{n_a} \times \frac{W}{n_i - s_i + 1} \tag{2}$$

Following the exploration phase, two additional terms are introduced to capture the global impact of a specific action up to that point. These terms are accompanied by the parameter

W, a weight that must be calibrated during the parameter tuning phase. Under the parameter W, the number of times the i-th node has failed to produce a positive outcome is recorded. Specifically, a value of one is added in cases where the node's number of visits equals its evaluation score.
In the case of microRTS, we decided to characterize each chosen action based on specific game-related information, such as:

- The unit's coordinates;
- The type of unit;
- The amount of resources carried;
- The unit's health points.

This approach enables the creation of a unique key for each type of action, based on the distinguishing information.
During the backpropagation phase, the global map is updated with the history of all actions, ensuring a consistently accurate representation of the situation.

2.3 Tree reuse

Tree Reuse [5] is a significant enhancement to the Monte Carlo Tree Search (MCTS) algorithm that focuses on leveraging previously constructed search trees across consecutive decision-making cycles. This technique aims to improve computational efficiency and reduce redundant calculations by retaining and utilizing the search tree developed in earlier iterations.
In traditional MCTS, a new search tree is constructed from scratch for each decision cycle. This approach can lead to a significant computational burden, particularly in real-time environments like microRTS. Tree Reuse addresses this inefficiency by preserving the portion of the search tree that remains relevant in subsequent cycles. Specifically, the sub-tree rooted at the node corresponding to the current game state is retained, while branches that are no longer valid are pruned.
In microRTS, careful consideration is required when structuring the search tree to accurately identify the new node that will serve as the root. As discussed in Section 2.1, each node in the tree represents actions for only one player. Consequently, to implement this enhancement, we opted to retain the branch of the tree corresponding to the action chosen for our units during each iteration. Then, in the subsequent iteration, we determine the node that represents the opponent's selected moves and designate the appropriate child node as the root of the new tree.

2.4 AWLM heuristic

One of the issues we identified while studying the *NaiveMCTS* implementation in microRTS is that, due to the limited time available for tree exploration, it often resorts to basic actions, such as deploying new workers. As a result, bases consistently have insufficient resources, which prevents the construction of key buildings like barracks. This becomes particularly problematic on larger maps, where barracks and fighting units are essential for victory. To address this, we developed a new heuristic, Against Worker in Large Maps (AWLM), which devalues states with a high number of workers. Specifically, this heuristic is activated when the distance between the player's bases is large, typically on larger maps.

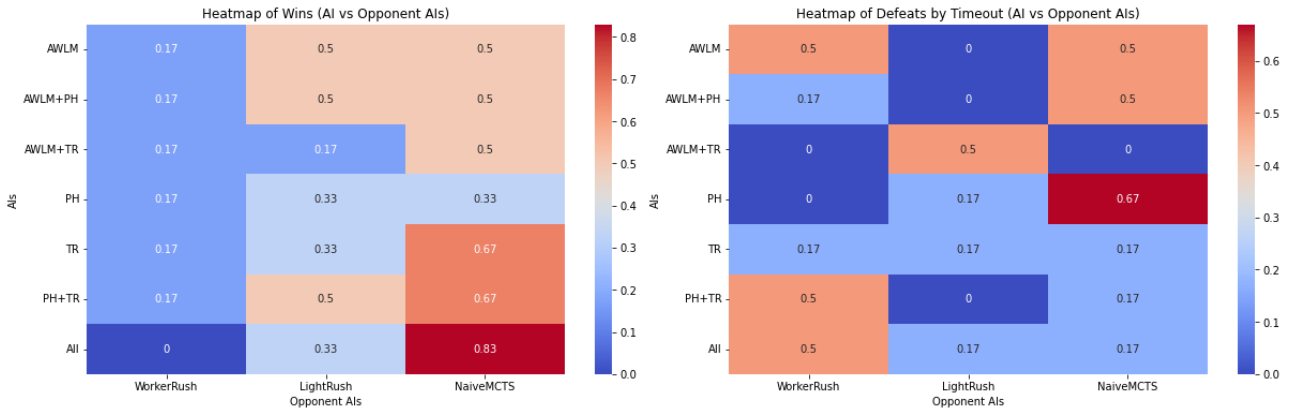
2.5 Newly created AIs

In conclusion, by combining the discussed enhancements inside the NaiveMCTS, we were able to develop 7 new different AIs:

- **PH_FusionRTS**: considering only the *Progresive History* enhancement;
- **TR_FusionRTS**: considering instead the *Tree Reuse* enhancement;
- **AWLM_FusionRTS**: considering only the new AWLM heuristic;
- **PH+TR_FusionRTS**: considering both *Progresive History* and *Tree Reuse*;
- **PH+AWLM_FusionRTS**: considering both *Progresive History* and the AWLM heuristic;
- **TR+AWLM_FusionRTS**: considering both *Tree Reuse* and the AWLM heuristic;
- **All_FusionRTS**: considering all the three enhancements together.

3 Results

In this section, we discuss the results obtained from analyzing our AIs after their development. As mentioned earlier, we organized a small fixed-opponent tournament where the AIs described in the previous section competed against three opponent AIs: *WorkerRush*, *LightRush*, and *NaiveMCTS*. Additionally, we used six different maps of varying sizes, ranging from 4x4 to 24x24. Our primary focus was on the matches against *NaiveMCTS*, as these provide the most accurate measure of the impact of our modifications.
In Figure 2, two heatmaps are presented, illustrating key results from the tournament. Specifically, the last column of Figure 2a shows the winning percentages of our AIs against the baseline *NaiveMCTS*. Almost all AIs managed to achieve a comparable number of wins to the existing *NaiveMCTS*. Notably, the AI implementing all enhancements attained the highest winning rate of 83%.
Additionally, we included a heatmap displaying the percentage of defeats our AIs experienced due to timeouts. In microRTS, an AI is automatically defeated if it exceeds the maximum time allowed to return an action. The high rate of timeout-related defeats suggests that further performance improvements could be achieved by fine-tuning certain parameters. This observation may also explain why AIs with fewer enhancements perform better against WorkerRush and LightRush: with fewer computations required, they reduce the likelihood of timeout-related defeats.
We also tried to visualize the percentages of win of our AIs when changing the size of the map. As it is possible to observe from the Table 1, we were able to achieve the higher number of wins with smaller size of the maps against the *LightRush* and *WorkerRush*, while we struggled on larger maps. This is expected, as the above mentioned AIs are able to early attack on those maps while MCTS AIs are not able to properly explore the search tree.
Finally, we conducted a focused analysis on the most frequently visited actions during the computation of the *Progressive History* enhancement. Specifically, Figure 3 illustrates the number of times the action for constructing a barracks was selected by the *PH+AWLM_FusionRTS* AI when playing against *WorkerRush* on various map sizes. Interestingly, this number appears to increase with map size but begins to decline after 12x12. This aligns with the expected behavior of AWLM, which encourages barracks construction on larger maps, while also reflecting the fact that *WorkerRush* becomes more effective at defeating our AIs as map size increases.



(a) *Winning percentage* (b) *Defeat by timeout percentage*

Figure 2: *Results achieved on a fixed opponent tournaments.*

Map	VS WorkerRush	VS LightRush	VS NaiveMCTS
4x4	86%	100%	86%
8x8	0%	86%	43%
10x10	0%	43%	57%
12x12	0%	0%	29%
16x16	0%	0%	57%
24x24	0%	0%	71%

Table 1: Percentage of wins against the opponent AIs when changing the map size in the tournament.

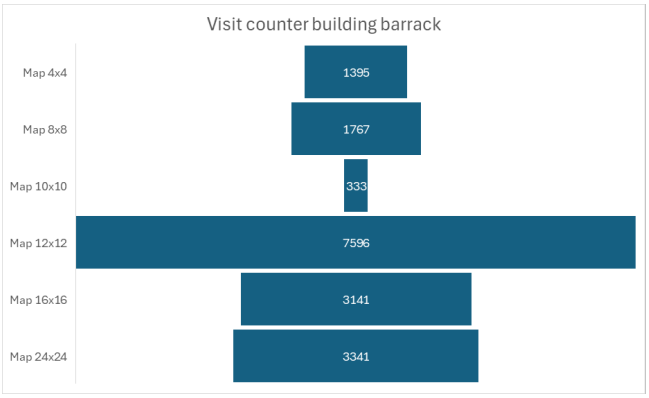


Figure 3: Number of times the action representing the construction of a barrack has been visited among several maps.

4 Conclusion

FusionRTS represents a concerted effort to advance AI capabilities in microRTS by addressing the limitations of existing MCTS implementations. Through the integration of Progressive History, Tree Reuse and a new heuristic, the project aims to develop an AI that is both strategically robust and computationally efficient. The tournament results served as strong validation of our methodologies showing a significant improvements with respect to the NaiveMCTS when combining multiple enhancements. However, further fine-tuning of certain hyperparameters could still lead to significant performance improvements.

References

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

[2] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers, editors, *Computers and Games*, pages 72–83, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

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

[4] J. A. M. Nijssen and M. H. M. Winands. Enhancements for multi-player monte-carlo tree search. In H. J. van den Herik, H. Iida, and A. Plaat, editors, *Computers and Games (CG 2010)*, volume 6515 of *Lecture Notes in Computer Science (LNCS)*, pages 238–249. Springer Berlin Heidelberg, 2011.

[5] Dennis J. N. J. Soemers, Chiara F. Sironi, Torsten Schuster, and Mark H. M. Winands. Enhancements for real-time monte-carlo tree search in general video game playing. *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2016.