Turn Action Extractions into Game Agent Module Decisions

Damijan Novak, Iztok Fister ml.

Institute of Informatics, University of Maribor, Koroška Cesta 46, 2000 Maribor, Slovenia
damijan.novak@um.si, iztok.fister1@um.si

Abstract. *In this article, the research in Real-Time Strategy games is made by utilizing Association Rule Mining to extract relevant opponent actions, and use those extractions as viable information input in game agent modules. The Self-Preservation Module is presented, which supports a game agent when dealing with game units on the game field. The module is positioned at the reactive level of the agent framework. Its main purpose is to constantly collect game world information and purify, extract, and organize it to provide instant threat assessments regarding the units under the game agents' control. The data gathered show distinctive opponent action variations, valuable in module decision-making.*

1 Introduction

During the last two decades, Real-Time Strategy (RTS) games have become one of the best testbeds for research of Artificial Intelligence (AI) in games research [1]. The main reason for this growth is that RTS games offer plenty of challenges for researchers, and there were also some tremendous commercial RTS game successes (e.g., StarCraft™ [2]). The genre requires many different areas of expertise, which the players must consider if they want to beat the opponent successfully. To cope with the complexity and sometimes partial information of the game environment [3], the computer-operated agent (a game agent) can be built modularly (i.e., decomposing problems into more minor separate problems) [4]. Each module is in charge of a specific task, a range of similar tasks, or a whole discipline. So far, a great deal of research has gone into trying to improve particular parts (e.g., aspects such as resource management, etc.) or the game as a whole (e.g., integrating the modules into higher-level architecture [5]) [6]. Especially, there was much interest in improving strategical and tactical planning, since they have the most significant impact on the game's result.

One of the current research trends is to bring some human elements into the RTS AI world (e.g., to mimic the human ability to adapt to previously unseen or unknown game situations [7]). Game agents can incorporate human-like factors such as creativity, adaptability, surprise tactics, activity levels, strategic planning, intuition, and perhaps most fundamentally, for all living beings, the will to survive. That way, the agents could be more realistic, and adaptive, leading to a more immersive and engaging player experience.

Our first contribution is the Self-Preservation Module (SPM), designed to instill a sense of self-preservation in game units. The SPM system analyzes the current game state and makes strategic decisions to keep units safe, emulating the human instinct for self-preservation. Some attempts can be found in the literature to incorporate these game mechanics in an agent, as seen in the allocation strategies that factor in elements like unit health and the balance between allies and enemies, which can be tied to the concept of self-preservation [8]. However, our approach is more comprehensive.

To ensure the modules' positive impact alongside well-established components of the RTS AI framework, and, in the process, not interfere with any of the components already in place, the proposal of the module is made to enable non-invasive decisions and control over self-preservation behavior for all units (i.e., keeping them out of harm's way, when in danger, if it is not interfering with the game agents' agendas) under the game agents' command.

Our second contribution is to make module decisions online adaptive by incorporating opponent action extractions achieved through Numerical Association Rule Mining (NARM) [9]. This innovative approach enhances the module's ability to adapt to changing game situations.

The rest of this paper is organized as follows: Section 2 delves into online adaptivity, exploring how systems can adjust and evolve in real-time environments. Section 3 outlines the SPM module architecture, with every constituting part of the module described in detail. In Section 4, the focus shifts to action extraction driven module choices, examining the process and its application in game agents' decision-making. Finally, Section 5 concludes the paper.

2 Online adaptivity

The ability of AI in games to adapt to changing environments is a significant aspect of RTS game research. The adaptivity field is divided mainly into offline and online adaptation. We talk about offline adaptation when game data (relevant game observation information like opponent actions, etc.) are collected during the gameplay, but the information it holds is not assessed until after the game. It can be processed after a game has been completed, or when the game environment is loaded next time. Offline adaptation is usually a good choice for processes requiring many computer resources (e.g., large amounts of data and/or computationally heavy algorithms). Therefore, they cannot be run during the game. The second field, online adaptivity, to which our work also belongs, is used for algorithms, which can process input into output in almost real-time. That way, the proper game actions can be taken while they still count. For example, if a friendly unit is attacked, it must decide either to run or fight back. It will be destroyed if it considers the response too long and stays still. Its loss will be in vain.

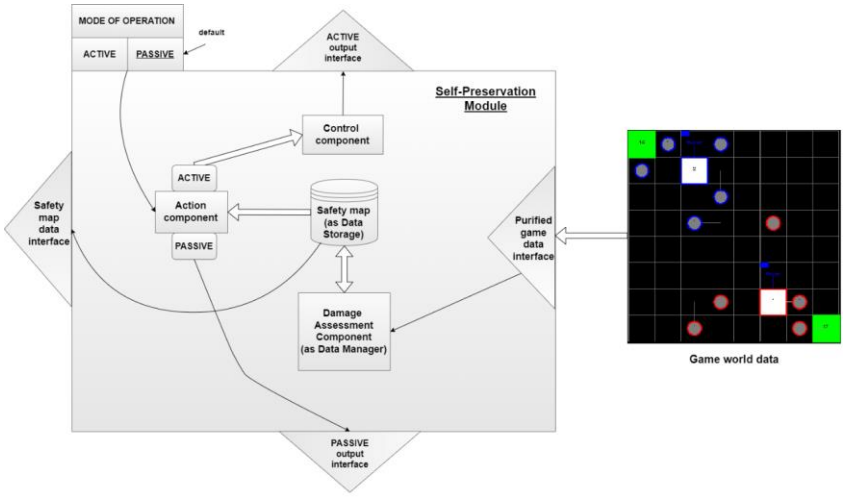

Figure 1. Self-Preservation Module architecture.

3 Self-Preservation Module architecture

The SPM is a background worker, which works on the reactive level of the game agents` framework. It can bring some advantage over the opponents, which would be overlooked otherwise. Its role is to gather information from the game world and assess the threats for each unit. If necessary, it can influence the game actively, but only if the actions comply with the goals of other modules (i.e., with game agents' permission), and do not dramatically affect the current strategy in place.

By default, its role is passive. This mode of operation does not have direct command over the units, and cannot impact the other modules. However, if the need arises, the strategic part of the game agent can set SPM to be active and grant it control over units. By this, the SPM can issue orders directly to the units that it protects, and decides appropriate actions on its own.

SPM was designed with simplicity in mind (Figure 1). The central component of the SPM is the Safety map. All other components are connected to it directly or indirectly. The Safety map contains a purified version of the current game world state and some relevant historical information about the past events, necessary to evaluate the threats and safety of the units.

The current operation mode and the involvement of SPM in a game are defined with a state variable called Mode of operation. As described above, there are two modes of behavior: Active and passive modes. Passive mode is the default behavior. Active mode can only be set from a higher module in the game agents` hierarchy. Besides that, there are three other SPM components: Damage Assessment Component (DAC), Action Component (AC), and Control Component (CC). CC is only operational when the SPM mode of operation is set to active. SPM communicates with other modules through four well-defined interfaces: Purified Game Data Interface (PGDI), Passive Output Interface (POI), Active Output Interface (AOI), and a Safety Map Data Interface (SMDI). PGDI is an input-only interface, while the other three are output-only interfaces.

SPM components initialize with the game agent. In this phase, three live game state-input parameters are essential: The width and height of the map (provided by the game engine's initial game state), the (ID) number of the player for which the SPM looks over its units and the (ID) number of the opponent. SPM then waits until its update method is triggered. By default, the update method is triggered between every frame (that's when the game agent has a time slice to process its moves), but, if need be, it can be delayed (every second, third, etc. frame). The update method only needs the current game state. During each SPM update, PGDI first extracts relevant feature information (e.g., damage done to friendly units in the last frame) and passes it to the DAC. DAC updates the state of the Safety map as presented with the pseudo-code in Algorithm 1.

Algorithm 1: Pseudo-code of DAC update procedure of the Safety map

```
    // Initial parameter settings:
    // - color and state parameter at which point we divide cells
    // - color and state parameter at which point we destroy cells
    // - unit destroyed parameter
    // - unit was hit parameter
    // - opponent behavior changed parameter
    // - decremental change parameter with every update cycle
    // - minimal cell size

    // Tree root and purified game data info from interface
1.  select cell, pgdi
2.  updateTreeRateOfColorStateDrop(cell)
3.  divideAllCellsThatNeedDivision(cell)
4.  destroyAllCellsThatAreNoLongerNeeded(cell)
5.  if pgdi.sizeOfUnitsCreated() > 0 then
6.    updateTreeForUnitsCreated(cell, pgdi)
7.  end if
8.  updateNeeded = false
      // update tree when units are destroyed
9.  if pgdi.sizeOfUnitsDestroyed() > 0 then
10.   updateTreeForUnitsDestroyed(cell, pgdi)
11.   updateNeeded = true
12. end if
      // update tree when units are damaged
13. if pgdi.sizeOfUnitsDamaged() > 0 then
14.   updateTreeForUnitsDamaged(cell, pgdi)
15.   updateNeeded = true
16. end if
      // update tree when opponent action extraction
      // behavior changed
17. if pgdi.opponentBehaviorChanged() = true then
18.   updateTreeForOpponentBehaviorChanged(cell, pgdi)
19. end if
      // update tree when units move
20. if cell.listOfCells.size() > 0 then
      updateTreeWhenUnitsMove(cell)
21. end if
      // update tree for bringing lowest color bottom - up
22. if cell.listOfCells.size() > 0 and updateNeeded then
23.   updateTreeToBringLowestColorUp(cell)
24. end if
```

AC then uses a Safety map to decide on required unit safety actions. For every action that must take place, a message is created. The message offers encapsulation of the following data: Which unit is in danger, the threat level, the cell in which the unit is positioned (i.e., a cell from a game tree, which is saved in the Safety map), and the position vector to where the unit should move to. If the default passive mode is activated, a list of messages gets passed to POI, otherwise to AOI.

The opponent action extraction process is made in the PGDI component of the SPM, and the data are pushed to the DAC. The DAC updates the Safety Map by considering many factors (e.g., a unit was damaged), including the action extraction data (blue colored lines 17 – 19 in Algorithm 1 signal where the opponent behavior changes are used to update the tree). The Safety map as data storage is designed dynamic as the quad tree (a tree data structure with up to four children) seen in Figure 2, so we only use memory when necessary. Meaning that, instead of designing a memory overview of a battle game map in a grid-like structure, a dynamic design allocation fashion is made of only »reserving a cell when units are present in the cell«.

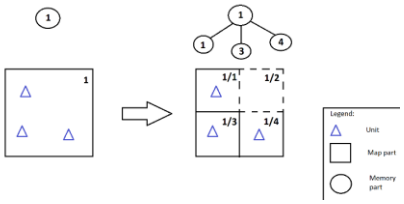

Figure 2. Quad tree structure.

The DAC continuously updates the quad-tree following the initial parameter settings in Algorithm 1 pseudocode. For example, if DAC requires greater granulation, the cell can be divided further, but only to the minimally set cell size. The cell can be divided when the units belong to more than one region (as seen in Figure 2).

The AC then has to decide if any friendly unit's color (i.e., status) has changed to dangerous (i.e., the unit is in danger). The color of each cell changes constantly. Color threat levels are depicted in Figure 3. The change in color towards black is triggered by increasing danger. Increasing danger happens when a friendly unit is damaged or destroyed, or when an opponent has changed action tactics towards attacks. The change towards white happens gradually with the passing of time.

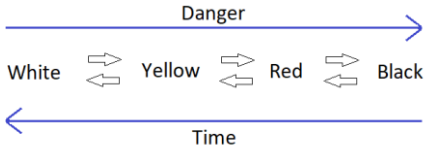

Figure 3. Color threat levels.

4 Action extraction driven module choices

In this section, the concept of action extraction as it pertains to game agents' decision-making processes is explored. First, the action extraction process is explained. Second, the experiment tests the feasibility of using action extractions in game agents' decision-making processes.

4.1 Action extraction process

The important function of SPM is to react to the opponent's strategy derived from the game states and actions taken so far in online mode (i.e., while the game is being actively played and the opponent's actions choices are imperative for the gameplay). The opponent's behavior can be non-deterministic; the same game-state can generate different reactions at different times. Only some of the moves have a significant impact on the game. For useful information extraction, the Numerical Association Rule Mining (NARM) method was used [10]. Association Rule Mining (ARM) is a data mining technique that finds data patterns representing relationships between items. ARM operates with binary data; NARM can also operate with numerical data.

The association rules were derived based on the gaming state and actions recorded from the beginning of the game. Only the association rules that contain the opponent's action and high-enough confidence were kept. Then, each action's frequency (the number of occurrences) is calculated. The result is a support vector of each possible (valid) opponent's actions.

4.2 Experiment: Action extraction driven module choices

The experiment was designed in a microRTS simulation environment [11]. The microRTS environment was designed with rules that mimic fully-fledged RTS games, but are of lower dimensionality (e.g., instead of the unit being able to move with pixel or coordinate system precision, it can only move up, down, left, or right for one cell at a time). In our case, its purpose is to provide the live game data for the NARM action extraction process. Features gathered from the microRTS environment and used with the NARM action extraction process were: The number of friendly / enemy workers / light units / heavy units / ranged units, a flag if a friendly base has been threatened, friendly / enemy resources left, number of friendly / enemy bases, and the number of enemy barracks. The actions used were: No action taken (0), move (1), harvest (2), unit returns (3), produce (4), and attack location (5). The opponent chosen for the study (i.e., its data during gameplay were recorded), was the UCT (note: The game agent is a part of the microRTS package) with default parameters set. The UCT agent played against the basic built-in RandomAI.

The data used for NARM were gathered from the game states across the span of the whole game. The mode of operations is such that, in each game state, the processing set holds the feature values and opponent actions of current and of all the game states before it. In each frame, the set is sent to NARM for processing. The threshold for the rules from which the actions are extracted is, for case study purposes, set to 0.5 (inclusive of this value). After the NARM process is

completed, and the action extraction of the opponent is made, the SPM can act on the received information.

Figure 4 shows the graph of the recorded data of a UCT agent during gameplay throughout the whole game. The abscissa axis indicates the consecutive frame numbers, while the ordinate axis shows the percentage distributions of each action for that specific frame.

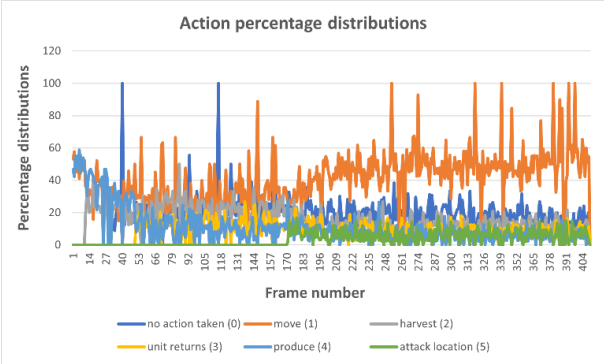

Figure 4. Graph showing the action percentage distributions across the whole game.

In Figure 5, only the percentage data of the action attack location (5) was isolated, to show better how the UCT agents' attack distribution changes with the passing game time. The changes between the highs and the lows provide the SPM with very informative data about when the opponent is switching game action usages, and if units are in heightened danger. Therefore, self-preservation measures are needed.

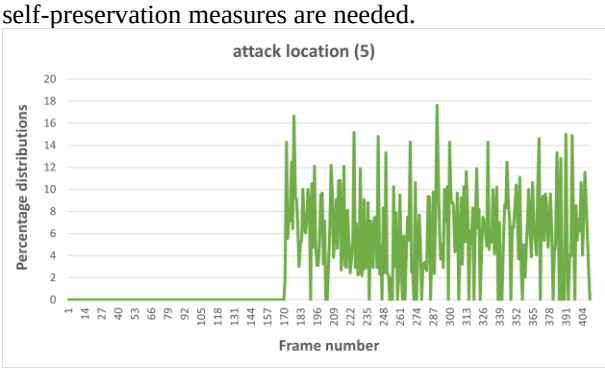

Figure 4. Graph showing the action percentage distribution of action attack location (5) across the whole game.

5 Conclusions

As proof of concept, the SPM was designed with three main goals. First, the module is universal, and can be included in every game agent using a modular design. Second, the module is simple enough to be implemented without much complexity and easily integrated into game agents. Third, it should not interfere with an active strategy in progress unless the game agent explicitly decides to use the module.

During initial testing of connecting the SPM with action extractions provided by NARM, it was noticed that, at the beginning of gameplay, the action extraction processing was possible in online mode, while, later on, when the sets of game features and actions stacked up, the game frame time slices (100 milliseconds) were exceeded, and the module operation resembled more that of the offline mode.

The data gathered on opponent behavior demonstrate a diverse range of action variations, as illustrated by the fluctuations in the two graphs representing agents' behavioral changes throughout the game. Overall, such distinct opponent action variations are crucial for module gameplay decision-making. With that in mind, future work will compare our approach with modern algorithms, such as the Policy Proximal Optimization (PPO) algorithm [12]. Using PPO, an evaluation could study the balance between self-preservation and other RTS game objectives. The feasibility of employing PPO for real-time control of specific game units or groups of units will also be examined.

Funding

This research was funded by the Slovenian Research Agency Research Core Funding No. P2-0057.

References

[1] M. Buro, "Real-Time Strategy Games: A New AI Research Challenge," in IJCAI'2003, Mexico: Morgan Kaufmann, pp. 1534–1535, 2003.

[2] M. J. Kim, K. J. Kim, S. Kim, and A. K. Dey, "Evaluation of starcraft artificial intelligence competition bots by experienced human players," In Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems, pp. 1915-1921, May, 2016.

[3] G. Synnaeve, and P. Bessiere, "Multiscale Bayesian modeling for RTS games: An application to StarCraft AI," IEEE Trans. Comput. Intell. AI Games, vol. 8(4), pp. 338-350, 2015.

[4] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "Survey of Real-Time Strategy Game AI research and competition in starcraft," IEEE Trans. Comput. Intell. AI Games, vol. 5(4), pp. 293-311, 2013.

[5] N. A. Barriga, M. Stanescu, and M. Buro, "Building placement optimization in real-time strategy games," In Tenth Artificial Intelligence and Interactive Digital Entertainment Conference, September, 2014.

[6] K. Adil, F. Jiang, S. Liu, W. Jifara, Z. Tian, and Y. Fu, "State-of-the-art and open challenges in RTS game-AI and Starcraft," Int. J. Adv. Comput. Sci. Appl, vol. 8(12), pp. 16-24, 2017.

[7] R. Lopes and R. Bidarra, "Adaptivity Challenges in Games and Simulations: A Survey", IEEE Trans. Comput. Intell. AI Games, 3(2), pp. 85-99, 2011.

[8] K. D. Rogers, and A. A. Skabar, "A micromanagement task allocation system for real-time strategy games", IEEE Trans. Comput. Intell. AI Games, 6(1), pp. 67-77, 2014.

[9] D. Novak, D. Verber, J. Dugonik, and I. Fister Jr. "Action-Based Digital Characterization of a Game Player," Mathematics, 11(5), 1243, 2023.

[10] M. Kaushik, R. Sharma, S. A. Peious, M. Shahin, S. Ben Yahia, and D. Draheim, "On the potential of numerical association rule mining," In International Conf. on Future Data and Security Engineering, Springer, Singapore, pp. 3-20, November, 2020.

[11] S. Ontanón, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," In Ninth Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 58 – 64, 2013.

[12] Y. Wang, H. He, and X. Tan, "Truly proximal policy optimization," In Uncertainty in Artificial Intelligence, PMLR, pp. 113-122, 2020.