Adaptive Online Opponent Game Policy Modeling with Association Rule Mining

Damijan Novak
Faculty of Electrical Engineering and Computer Science
University of Maribor
Maribor, Slovenia
damijan.novak@um.si

Iztok Fister Jr.
Faculty of Electrical Engineering and Computer Science
University of Maribor
Maribor, Slovenia
iztok.fister1@um.si

***Abstract*—Opponent modeling is a research aspect that needs close attention when facing an opponent in a complex game environment. Real-Time Strategy (RTS) games are a representative of one of the highest complex game environments. In RTS games, the players` tactical and strategical decisions need constant adaptation in order to win the game. In this work, a preliminary study is reported on opponent modeling in RTS games via the incorporation of association rule mining to identify game policies in online mode. Such insight into opponents' mode of operation should provide a player with vital information regarding choosing advantageous gameplay countermeasures against an opponent.**

Keywords—Association Rule Mining, Artificial Intelligence in games, Numerical Association Rule Mining, opponent policy modeling, Real-Time Strategy games, uARMSolver

I. Introduction

Digital games are a great form of entertainment for many players. Due to the variety of games, different categorizations are used to group them accordingly. They can be categorized by genres (genres such as Massive Multiplayer Online games [1], 2D platform games [2], or survival-based games [3]), development models (e.g. indie games [4]), domains (e.g. serious games in education and training [5]), computational complexities (e.g. Multi-player Online Battle Arena [6] with enormous state-action space), to name the few. One of the game genre games at the top of the list of computational complexity [7] is Real-Time Strategy (RTS) games.

RTS games are a genre of video games where players have to master gameplay aspects such as production and resource management (game-specific units and structures can be made with careful utilization of resources), gameplay decisions (ranging from high strategical command across tactical decisions all the way to the low-level unit behavior), scouting (of opponent units), and sometimes even diplomacy, to gain an edge advantage in the game, which would result in victory for the player. So, for the player to be able to defeat the opponent, constant overwatch must be made regarding what the opponent is doing (opponent modeling [8]), with adjustments to their own gameplay made accordingly.

The main research idea regarding opponent modeling is to use the Machine Learning (ML) approach, more specifically Association Rule Mining (ARM), to perform a search for rules on the RTS data gathered during the online gameplay (i.e. while the game is being played). The action part of the rules is then matched to the pattern, where the patterns are represented with predefined game policies. By achieving a successful pattern match, a conclusion can be made regarding which game policy is used by the opponent, and proper adjustments and countermeasures can be executed.

The paper is structured as follows. In the second section the main definitions for the game blocks needed in the method are identified in the related works. Section three discusses RTS game subdomains important to the research regarding opponent modeling. Section four presents Numerical Association Rule Mining (NARM). The motivation, design and limitations of the opponent game policy modeling method are provided in section five. The method is proposed in section six. In section seven the method design is tested with the experiment and the results are provided. Section eight provides the discussion. The article concludes in section nine.

II. Game blocks identified in related works

In this section the main game blocks are presented which are required in the concept of method, and which were identified through the related works. Specifically, the focus is on the rule mining, pattern, feature and replays inside the game domain.

In Leopold et al. [9] it was noted that rule mining typically generates a considerable number of rules, from which only rules meeting a certain support threshold are kept. Therefore, the need was identified for the threshold to be included into this research.

In Li et al. [10] a loose definition of a pattern was provided in this quote: "There are no fixed rules of what constitutes a "pattern" in gameplay, and thus any dimension (e.g., time, position, events) or a combination of multiple dimensions can contain the factor(s) that define(s) a pattern behaviors.". They also wrote that existing data-driven game analysis methods depend on data experts to construct a representative feature space with external knowledge manually.

In Bosc et al. [11] they represented a pattern as a frequent series of game actions done by the two players extracted from a large database of replay games. They also defined replay as a match record that contains all actions generated by the players.

III. Real-Time Strategy Game subdomains

A short theoretical background of RTS game subdomains relevant to this work is provided in this section. First, a short description is given of some of the key aspects regarding the RTS games, especially how they differentiate from the classical board games upon which they are based. In this segment the microRTS [12] is also described, which is a simulation environment designed for rapid testing of new research ideas. The environment will be used in the experiment to gather data on the RTS features. Next, descriptions are provided regarding features and their connection to the rules. Rules are basic components upon which the opponent modeling is based, and for which the description is also given. The section concludes with game policy explanations.

A. Key aspects of a Real-Time Strategy games and microRTS simulation environment

RTS games somewhat share the principle of their turn-based board game cousins regarding the action execution, but the actual implementation differs in a few important key points. In the turn-based board games (e.g. chess) actions upon the game environment (e.g. the chess board) are executed in turns (e.g. after the first player made a move/action the second player has time to make his/her move). This is the same for the RTS games, but with the important difference of a real-time component. The actions can be executed upon every new game state, which, in RTS games, usually happens 24 [13] or 30 [14] times per second (ca. 30 milliseconds between a change of game states). Therefore, the game is played out in real-time, where, for the human player, (who observes the game on a second-time scale), it looks like the actions and their results showing in the environment are instant (seamless progression of the game).

The second important difference is that both players can execute their actions on every game state (simultaneous moves), as opposed to the turn-based board games, where the first player executes a move on the game board, followed by the move of the second player upon the new/updated game board.

The third difference is that there is not only one action made per player and per game state, but each player can execute multiple actions per every game state. In fact, the player can execute an action for every unit (called unit-action) in his/her possession present in the game environment. A combined group of all the unit-actions is presented in the form of player-actions [15].

The microRTS supports all the above-mentioned key aspects, as well as some others, such as: Durative actions (i.e. they are not instant, so they can last over multiple game states), various sizes of game maps, support for multiple players, options to make the game environment partially observable and non-deterministic, etc. It supports production of units and structures, resource gathering, and the support for battle game mechanics. Its simplicity comes on account of the small number of supported units (one type of resource gathering unit called a worker, which can also attack, and three purely battle oriented units: Light, heavy, ranged) and structures (two structures, named base and barracks), unit move capabilities (up, down, left, and right), the small number of unit parameter types, 2D only environment, and the like. However, although the microRTS environment is kept simple, the full RTS complexity is retained.

B. Features and rules

Some of the commonly found features in RTS games are unit type, health and terrain/map [16]. In a typical RTS game the number of features can surpass hundreds [17]. Basic features (e.g. microRTS's unit types [18]) are usually accessed directly through the methods found in the game engine, like those connected to the game state. The game state can also hold extended feature information, like which durative actions units on the game map are currently executing.

Commercial games usually address specific RTS game aspects (e.g. tactical battlefield-arrangements) with the help of basic feature information extraction from the game engine [19]. Extraction can be made by hand, and stored in the form of a rule, where the rule is defined as the regular continuous aspect of the game [20], with different recombinations of rules forming a (dynamic) script [21] or pattern [22]. Such constructs can then be used to create opponents [23], as well as to model opponents [24].

C. Opponent modeling

Playing the RTS game is more than just following a strategy agenda set in a straight line. One must also be cautious about the surrounding environment, especially of every action performed by the opponent, so that corrections can be made of the tactics and alterations in the general strategy. Anticipating what the opponent will or can do through opponent modeling is, therefore, a requirement for a good game performance [25].

If an assumption is made that an RTS game agent implements strategic and tactical command [26] and can recognize opponents' intentions [27], synergy of these two agents` mechanics should result in a better performance. Meaning, an agent can change tactics on the fly, and the strategies can be changed [28] or enhanced (e.g. enhancing scripted behavior through a look-ahead search [29]).

Since the RTS games are the most complex game genre environments [7], the possibility of finding all the possible strategies at a given time is intractable. But, given that RTS games are Markov games [30], it is often possible to learn a set of policies that encapsulate the diversity of possible strategies [31]. To investigate players and to create such policies, replay logs of the already played games (e.g. games of professional players) may be a beneficial method [32].

D. Game policy

Designing a general game policy for the RTS games is not an easy task, due to the multiple simultaneous aspects of the game that must be considered. Aspects such as resource gathering, battles, production of new structures and buildings, or research tree extensions. Therefore, to be able to cope with the problem of managing all those aspects, the researchers have to divide the problem into subproblems, based on the planning timescale (strategy, tactics and reactive control), and by the subproblem`s function (production and resource management, opponent modeling, scouting operations, and others) [33].

Static game policies that are predefined (scripted) in the game agents are obviously not as good a choice as the dynamic ones. However, their advantage is speed, because they are vulnerable against search-based methods [34]. The game agents therefore depend on a good design of game policy, which will be robust enough to help deal with seemingly similar game situations, unpredictable, so that the opponent will have a hard time figuring out its plans, fast and dynamic in a sense of offering variability of operations in the same game scenarios. Game policy can also be dependent on the state-evaluation and state-forwarding functions [35] (the usage of different kinds of models, e.g. combat models [36]).

IV. Numerical Association Rule Mining

ARM stands for an important ML method whose task is to find relationships between attributes in a transaction database [37]. These relationships are presented as implications, where the left side of the mined rule represents an antecedent and the right side the consequent. The roots of ARM go back to the early 90s, when the first algorithm named Apriori was proposed.

NARM is considered as a variation of ARM [38][39]. The main difference between NARM and ARM lies in handling numerical attributes, where, in NARM, numerical attributes are handled without discretization. It means that NARM algorithms operate directly with both categorical, as well as numerical, attributes. Typically, discretization introduces certain problems which result in the loss of information and noise entering into data. Thus, NARM algorithms which do not need discretized data as an input provide more exact mined rules. Nowadays, most of the NARM algorithms are based on stochastic population-based nature-inspired algorithms due to their ability for searching the very big search space.

uARMSolver (universal ARM Solver) is a novel software tool for tackling the NARM [40]. Software is written fully in C++ and runs on all platforms. uARMSolver covers all stages of the ARM process, from data preprocessing, rule mining and visualization of results. Many NARM algorithms are included in this tool, e.g. algorithms based on Differential Evolution or Firefly algorithms.

V. Motivation, Design and Limitations of the Proposed Opponent Game Policy Modeling Method

The main idea is to record every action that the opponent has done in the game environment, along with all the relevant game feature information during the game playout. This information is saved, because the past holds important clues which can reveal the gameplay patterns of the opponent. By recording the opponent, mining for rules in such a recording and creating a pattern upon the rules, which results in finding the most probable game policy, can provide an advantage in the game. For example, if, at the beginning of the game, the game agent`s game policy selects advanced production, which provides an advantage in the middle game, but the opponent plans an early rush tactic which would destroy the game agent`s base before the game really even starts, and a discovery of the planned opponents rushing game policy is made, the game agent can immediately execute countermeasures (e.g. switching the tactics to building an early defense).

The design is also driven by the choice to have a method capable of providing results without the need for the forward model, i.e. no simulations into the possible future game states. Every result is, therefore, derived from the history of past states (replay of the opponent's gameplay actions). Such method could prove its worth where there is no access to the future model (e.g. the game engine/environment doesn't support such calls, or there is no viable solution to simulate future states), or where the game agent must only operate at runtime / online mode (no pre-processing available).

The method is also implemented to be time adjustable, to enable the possibility of discovering what patterns the opponent is using in a chosen time frame. A time frame component is important, because by setting it to the nearby time (i.e. the last half minute of game time), patterns regarding recent tactics can be discovered, and by setting a longer period (e.g. the whole game from the beginning to the current game frame), extrapolation of tactics (or even strategies) that are being used over a longer period of time are possible.

There are some limitations or shortcomings regarding this chosen design. The first obvious one is that, if the opponent changes his/her patterns of gameplay during the game, this will only be sensed in the next state (or the next few states) when new information is processed and new patterns are discovered. No future predictions are made at this point.

A current shortcoming is that automatic RTS feature identification and extraction are not implemented yet. The first feature identification is made by hand, but possible improvements are planned for future research articles (e.g. feature extraction as the preprocessing step [41]).

VI. Concept of Method

The method for opponent game policy modeling with ARM (ogpmARM) (Fig. 1) is proposed and described in the following steps:

1. Define:
    - The relevant game features that are recorded from the game state during the gameplay.
    - The confidence value threshold θcv above which the rules provided by uARMSolver will be kept.
    - The maximum passed time for which the rules are kept. It can also be expressed in the form of the value of passed game state iterations.
2. Whenever the opponent executes the player-action, the values for chosen features from the game state are recorded, along with the action-type (e.g. move, attack, produce, and other actions). One line of the data record is created for each unit-action found in the player-action set. The record is only made during a set time interval. Previous saved records are deleted. The closed time interval during which the records are kept is [a - mpt, a], where a is the result of a function call time(current_frame_num), and mpt is the maximum passed time for which the rules are kept. If the start of the interval is under 0, or if the mpt is set to -1, the interval starts with the first game frame (beginning of the game).

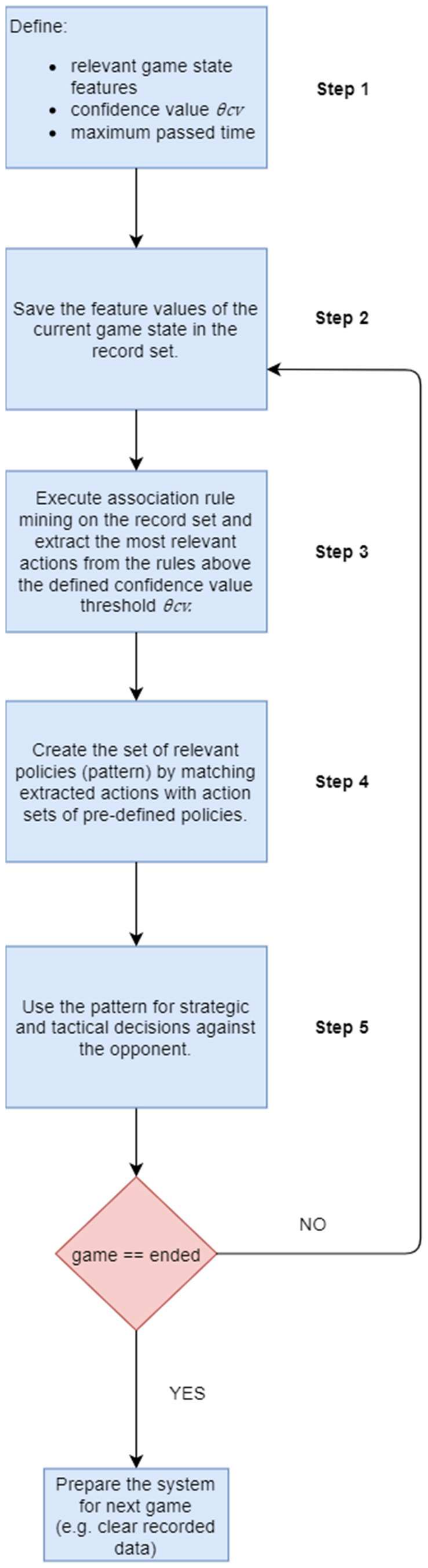
Define:
relevant game state features
confidence value θcv
maximum passed time
Step 1
Save the feature values of the current game state in the record set.
Step 2
Execute association rule mining on the record set and extract the most relevant actions from the rules above the defined confidence value threshold θcv.
Step 3
Create the set of relevant policies (pattern) by matching extracted actions with action sets of pre-defined policies.
Step 4
Use the pattern for strategic and tactical decisions against the opponent.
Step 5
game == ended
NO
YES
Prepare the system for next game (e.g. clear recorded data)

Fig. 1. Method for opponent game policy modeling with ARM (ogpmARM).

3. The ARM is executed upon the gathered data set and the rules above the θcv are selected. The action-types are then extracted from the rules. That action-type represents the action which has the highest confidence for being the main action type that the opponent is using.
4. The policies where the extracted action type is present are selected from the pre-defined set of game policies. The set of chosen game policies forms a pattern.
5. Use the pattern for strategic and tactical decisions against the opponent.

VII. Experiment and Results

The microRTS [42] game simulation environment (used in IntelliJ IDEA 2020.3 tools with OpenJDK 11.0.5) was utilized for the purpose of data gathering.

The following relevant game state features are defined for which the values will be recorded: Number of friendly light, heavy and ranged units, number of friendly workers, number of opponent light, heavy and ranged units, number of opponent workers, flag if a friendly base is under threat (true / false), number of friendly and opponent resources left, number of friendly and opponent bases, number of friendly and opponent barracks. Every recorded line is then paired with every unit-action that the opponent executes.

To create the policy action sets, the following possible actions from which to choose (they are indexed by microRTS for faster processing) are first defined: No action taken (0), move (1), harvest (2), unit returns (3), produce (4), attack location (5). Second, three basic policy action sets are created from the set of possible actions:

- Resource gathering/harvesting policy: No action taken (0), harvest (2).
- Tactical policy: No action taken (0), move (1), unit returns (3), attack location (5).
- Production policy: No action taken (0), produce (4).

The unified action naming for the actions move and produce is used (i.e. moves up, down, left, right are represented uniformly as a move), but further decomposition of actions is also viable. Every basic policy holds the no action taken (0) to represent a choice (a unit can choose this option at any time).

The confidence value θcv was set to 0.5 and maximum passed time was set to -1 (so that all the records from the first game frame forward are kept). The maximum running game time was set to 3,000 frames. The uARMSolver and microRTS experimental environments were used with default values (i.e. as they are published on GitHub in their original form).

The experiment included three testing game agents: RandomBiasedAI, UCT and NaiveMCTS [12] [42]. The testing game agents played against the RandomAI agent on the pre-included microRTS map called basesWorkers16x16. All the game agents used are part of the microRTS environment package. Each of the game agents played the game three times. For each of the played games the extracted set of actions (i.e. the number of times the action appeared in the mined rule set) per different game agent is presented in Table I. The numbers inside each action set are indexed by the action index, and they show the intensity of the action being used. For example, the action set {5, 25, 8, 4, 8, 0} provides information that the action with index 0 has intensity of 5 (i.e. this action was extracted from the rules 5 times), the action with index 1 has an intensity of 25, and so on. The number of the action is set to 0 if it hasn't appeared in the rule set at least once. The information of intensity can be useful in steps 4 and 5 of the ogpmARM, as discussed in the next section.

The Table I columns are divided to show action sets after the ogpmARM is executed on the 500th, 1,000th, 1,500th, 2,000th, 2,500th and 3,000th milestone frame (milestone frames can be seen as a time-line where a specific sub-strategy would take place). Max. passed time is set to -1. The specific milestone frames are omitted from Table I if the game has finished before reaching them.

TABLE I. ACTIONS EXTRACTED IN STEP 3

	RandomBiasedAI	**UCT**	**NaiveMCTS**
1.	500th: {5, 25, 8, 4, 8, 0}	500th: {0, 28, 7, 5, 5, 0}	500th: {7, 27, 6, 6, 3, 0}
	1,000th: {7, 30, 6, 5, 5, 0}	1,000th: {10, 18, 5, 7, 3, 3}	1,000th: {9, 19, 9, 5, 0, 4}
	1,500th: {8, 16, 3, 4, 6, 1}	1,500th: {6, 17, 9, 5, 3, 2}	1,500th: {10, 21, 8, 3, 3, 2}
	2,000th: {3, 16, 8, 5, 6, 4}	2,000th: {0, 29, 4, 8, 3, 0}	/
	2,500th: {4, 27, 5, 4, 9, 1}	/	/
	3,000th: {4, 17, 9, 5, 6, 2}	/	/
2.	500th: {16, 16, 5, 8, 12, 0}	500th: {4, 19, 9, 7, 4, 0}	500th: {5, 25, 7, 6, 8, 0}
	1,000th: {7, 10, 4, 4, 4, 0}	1,000th: {5, 18, 5, 6, 2, 4}	1,000th: {8, 23, 7, 4, 5, 0}
	1,500th: {3, 21, 3, 3, 6, 4}	1,500th: {7, 15, 6, 5, 2, 1}	1,500th: {10, 21, 6, 3, 5, 6}
	2,000th: {9, 11, 3, 5, 7, 1}	2,000th: {8, 23, 5, 9, 1, 4}	/
	2,500th: {5, 17, 3, 2, 5, 3}	2,500th: {6, 28, 8, 6, 1, 3}	/
	3,000th: {10, 16, 4, 5, 7, 2}	/	/
3.	500th: {17, 12, 3, 6, 7, 0}	500th: {12, 25, 8, 8, 2, 0}	500th: {7, 18, 6, 5, 1, 0}
	1,000th: {19, 14, 5, 5, 8, 0}	1,000th: {5, 20, 6, 5, 2, 1}	1,000th: {5, 19, 4, 5, 6, 4}
	1,500th: {15, 18, 3, 7, 5, 0}	1,500th: {4, 16, 4, 6, 3, 5}	1,500th: {6, 18, 7, 5, 5, 3}
	2,000th: {9, 12, 3, 9, 2, 2}	2,000th: {7, 18, 9, 4, 2, 2}	2,000th: {8, 23, 3, 4, 4, 1}
	2,500th: {13, 14, 3, 5, 7, 4}	/	/
	3,000th: {7, 15, 6, 7, 4, 2}	/	/

VIII. DISCUSSION

From Table I it can be observed that all of the indexed actions have appeared in it, and, therefore, the patterns of all the basic policies are present. This observation is logical, since each of the tested game agents has the capability for the full-blown RTS game (i.e. resource gathering/harvesting, production and military operations). On a side note, if this information isn't known in advance, the steps from 1 to 3 could be helpful for identification of the level of sophistication of the game agent (e.g. if the unknown game agent needs to be tested, or the game agent which is still in the development phase performs as it should).

The information of no action taken (0) from Table I is disregarded, because this action is used in every game policy. However, its intensity could still provide the clues about the behavior of each game agent (e.g. overuse of it by a game agent at the beginning (500th) of the game, could indicate a lack of the agent`s directed strategical guidance).

There is a heavy presence of the action move (1) across all three agents and all the frames` measures. When combined with the action's unit returns (3), and attack location (5), the tactical (policy) pattern is formed (step 4), which is an important opponent behavior to look out and care for. It can be observed from Table I that this pattern never formed in the 500th frame (i.e. we did not have agents utilizing early rush/attack tactics). It only starts appearing in the 1,000th frame (and only UCT always utilized it until then). Agents who use tactical pattern early on are clearly offensive oriented. Step 5 of the ogpmARM would therefore create countermeasures like additional unit production and increased tactical policy usage (as the adage says: "The best defense is a good offense.").

NaiveMCTS data reveal interesting facts when step 4 of the ogpmARM is utilized. In all three games, at the beginning of the game, the agent utilizes a fair amount of resource gathering pattern (harvesting actions (2) for the 500th frame are: 1st game - 6x, 2nd game - 7x, 3rd game - 6x). In the 1st game the intensity of the production pattern is low (action produce(4)), but the tactical policy pattern is present. In the 2nd and 3rd games, the agent utilizes a fair amount of the production pattern, as well as incorporation of a tactical policy pattern, and both stay consistent across the middle and/or end games. This creates constant pressure on the opponent, and the game ends quickly afterwards (always in under the 2,000th frame).

IX. CONCLUSION

In this article the adaptive online ogpmARM is proposed, which, at its core, utilizes ARM. The preliminary observations of the data show that matching pre-defined policies with the actions acquired from the rule sets of the uARMSolver, provides useful game policies (patterns) when tested on three game agents. Countermeasures based on those findings could be valuable in the actual/commercial games, which are one of the focuses of our future research. However, because the commercial games are usually controlled tightly, which poses challenges to the interested researchers [32], the plan is to try testing the ogpmARM in free RTS game engines such as Spring [43] (some of the games built upon this engine are based on the commercial game Total Annihilation™), or take advantage of the Brood War Application Programming Interface (BWAPI) [44], which allows interaction with the popular commercial StarCraft™ game engine.

We would also like to take the future research direction of comparing our ogpmARM against other well-known Reinforcement Learning (RL) algorithms. One of such on-policy based RL algorithm groups that we would like to try are the Proximal Policy Optimization algorithms [45]. Additionally, we could also consider testing the ogpmARM on other types of strategy games (e.g. turn-based strategy games), where the players do not play simultaneously, and where there is more time for considering the optimal strategical/tactical countermeasures. Therefore, the Deep Learning RL methods could also be considered, such as Deep Q-Learning from demonstrations [46].

There is also much room for improvement of the ogpmARM in the form of automatic feature extraction, which would then be used in the rules` generation. The ogpmARM could also serve us as the previously unknown game policy (pattern) generator (i.e. combining the actions extracted from rules based on their intensity). Such generated policies could then be incorporated in the game agent, therefore removing the need for manual creation of them. This way we could also create multiple variations of the same game policy (e.g. policies where only one action differs) regarding the specific RTS aspect (e.g. different types of resource gathering).

The overall usefulness of the ogpmARM lies in its time management, where adjusting time interval during which the records are used while in-game (i.e. online learning), we can search for patterns regarding RTS micromanagement of units (shortest time span of the game), tactics (medium time span) or possibly even strategies (longest time span). Time management and avoiding the need of pre-processing large amount of the replays, could provide the edge advantage in highly complex environments of RTS game spaces.

Acknowledgment

The authors acknowledge the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0057).

References

[1] P. Jordan, W. Buente, P. A. Silva, and H. Rosenbaum, "Selling out the magic circle: free-to-play games and developer ethics," in Proceedings of 1st International Joint Conference of DiGRA and FDG, vol. 13, no. 1, pp. 1-16, 2016.

[2] G. M. Costa, and T. B. Borchartt, "Procedural terrain generator for platform games using Markov chain," in Proceedings of SBGames 2018, 2018.

[3] S. Saaidin, and M. Kassim, "Recommender System: Rating predictions of Steam Games Based on Genre and Topic Modelling," in 2020 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS), IEEE, pp. 212-218, 2020.

[4] S. Castree, "A Problem Old as Pong: Video Game Cloning and the Proper Bounds of Video Game Copyrights," available at SSRN 2322574, 2013.

[5] L. B. Çapli, "Serious game development methodology with system and human oriented approach," Master's thesis, Middle East Technical University, 2019.

[6] D. Ye, G. Chen, W. Zhang, S. Chen, B. Yuan, B. Liu et al, "Towards playing full moba games with deep reinforcement learning," Advances in Neural Information Processing Systems, vol. 33, 2020.

[7] G. Synnaeve, and P. Bessiere, "Multiscale Bayesian modeling for RTS games: An application to StarCraft AI," IEEE Transactions on Computational intelligence and AI in Games, vol. 8, no. 4, pp. 338-350, 2015.

[8] F. Schadd, S. Bakkes, and P. Spronck, "Opponent Modeling in Real-Time Strategy Games," in GAMEON, pp. 61-70, 2007.

[9] J. Leopold, I. Alobaidi, and N. Eloe, "Predictive Analysis of Real-Time Strategy Games Using Discriminative Subgraph Mining," in ICDM, pp. 176-190, 2019.

[10] Q. Li, Z. Wu, H. Qu, and X. Ma, "Co-Design of an Interactive Analytics System for Multiplayer Online Battle Arena Game Occurrences," Data Analytics Applications in Gaming and Entertainment, Auerbach Publications, pp. 247-276, 2019.

[11] G. Bosc, M. Kaytoue, C. Raïssi, and J. F. Boulicaut, "Strategic Patterns Discovery in RTS-games for E-Sport with Sequential Pattern Mining," in MLSA@ PKDD/ECML, pp. 11-20, 2013.

[12] S. Ontanón, "The combinatorial multi-armed bandit problem and its application to real-time strategy games," in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 9, no. 1, 2013.

[13] M. Stanescu, N. A. Barriga, and M. Buro, "Hierarchical Adversarial Search Applied to Real-Time Strategy Games", in Tenth Artificial Intelligence and Interactive Digital Entertainment Conference, 2014.

[14] M. Magnusson, and T. Hall, "Adaptive goal oriented action planning for RTS games", Independent thesis Basic level, Blekinge Institute of Technology, School of Computing, 2010.

[15] S. Ontanón, "Informed monte carlo tree search for real-time strategy games," in 2016 IEEE Conference on Computational Intelligence and Games, IEEE, pp. 1-8, 2016.

[16] N. A. Barriga, M. Stanescu, F. Besoain, and M. Buro, "Improving RTS game AI by supervised policy learning, tactical search, and deep reinforcement learning," IEEE Computational Intelligence Magazine, vol. 14, no. 3, pp. 8-18, 2019.

[17] Y. N. Ravari, S. Bakkes, and P. Spronck, "Playing Styles in StarCraft," in European GAME-ON Conference on Simulation and AI in Computer Games, 2018.

[18] S. Ontañón, N. A. Barriga, C. R. Silva, R. O. Moraes, and L. H. Lelis, "The first microrts artificial intelligence competition," AI Magazine, vol. 39, no. 1, pp.75-83, 2018.

[19] C. Si, Y. Pisan, and C. T. Tan, "A scouting strategy for real-time strategy games," in Proceedings of the 2014 Conference on Interactive Entertainment, pp. 1-8, 2014.

[20] M. Abbadi, F. Di Giacomo, R. Orsini, A. Plaat, P. Spronck, and G. Maggiore, "Resource entity action: A generalized design pattern for rts games," in International Conference on Computers and Games, Springer, pp. 244-256, 2013.

[21] H. J. Van den Herik, H. H. L. M. Donkers, and P. H. Spronck, "Opponent modelling and commercial games," in Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 15-25, 2005.

[22] P. Yang, B. E. Harrison, and D. L. Roberts, "Identifying patterns in combat that are predictive of success in MOBA games," in Proceedings of Foundations of Digital Games, 2014.

[23] M. Zohaib, "Dynamic difficulty adjustment (DDA) in computer games: A review," Advances in Human-Computer Interaction, 2018.

[24] H. Park, H. C. Cho, K. Lee, and K. J. Kim, "Prediction of early stage opponents strategy for StarCraft AI using scouting and machine learning," in Proceedings of the Workshop at SIGGRAPH Asia, pp. 7-12, 2012.

[25] J. Goodman, and S. Lucas, "Does it matter how well I know what you're thinking? Opponent Modelling in an RTS game," in 2020 IEEE Congress on Evolutionary Computation, IEEE, pp. 1-8, 2020.

[26] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "RTS AI Problems and Techniques," in Encyclopedia of Computer Graphics and Games, Springer, 2019.

[27] F. Kabanza, P. Bellefeuille, F. Bisson, A. R. Benaskeur, and H. Irandoust, "Opponent Behaviour Recognition for Real-Time Strategy Games," in Proceedings of the 5th AAAI Conference on Plan, Activity, and Intent Recognition, ser. AAAI Workshops, vol. 10, no. 5, 2010.

[28] B. D. King, "Adversarial planning by strategy switching in a real-time strategy game," Master's thesis, Oregon State University, 2012.

[29] N. Barriga, M. Stanescu, and M. Buro, "Puppet search: Enhancing scripted behavior by look-ahead search with applications to real-time strategy games," in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 11, no. 1, 2015.

[30] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in Machine learning proceedings 1994, Morgan Kaufmann, pp. 157-163, 1994.

[31] B. King, A. Fern, and J. Hostetler, “Adversarial Policy Switching with Application to RTS Games,” in Eighth Artificial Intelligence and Interactive Digital Entertainment Conference, vol. 8, no. 1, 2012.

[32] J. L. Hsieh, and C. T. Sun, “Building a player strategy model by analyzing replays of real-time strategy games,” in 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), pp. 3106-3111, 2008.

[33] A. Elogeel, “Selecting robust strategies in RTS games via concurrent plan augmentation,” Doctoral dissertation, 2015.

[34] D. Churchill, A. Saffidine, and M. Buro, “Fast heuristic search for RTS game combat scenarios”, in Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, vol. 8, no. 1, 2012.

[35] A. Uriarte, and S. Ontanón, “Game-tree search over high-level game states in RTS games,” in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 10, no. 1, 2014.

[36] A. Uriarte, and S. Ontanón, “Combat models for RTS games,” IEEE Transactions on Games, vol. 10, no. 1, pp. 29-41, 2017.

[37] I. Fister Jr., and I. Fister, “A brief overview of swarm intelligence-based algorithms for numerical association rule mining,” Applied Optimization and Swarm Intelligence, Springer, 2020.

[38] E. V. Altay, and B. Alatas, “Performance analysis of multi-objective artificial intelligence optimization algorithms in numerical association rule mining,” Journal of Ambient Intelligence and Humanized Computing, pp. 1-21, 2019.

[39] E. V. Altay, and B. Alatas, “Intelligent optimization algorithms for the problem of mining numerical association rules,” Physica A: Statistical Mechanics and its Applications, 540, 123142, 2020.

[40] I. Fister, and I. Fister Jr., “uARMSolver: A framework for Association Rule Mining,” arXiv preprint arXiv:2010.10884, 2020.

[41] H. T. Kim, and K. J. Kim, “Learning to recommend game contents for real-time strategy gamers,” in 2014 IEEE Conference on Computational Intelligence and Games, IEEE, pp. 1-8, 2014.

[42] Z. Yang, and S. Ontanon, “An Empirical Survey on Methods for Integrating Scripts into Adversarial Search for RTS Games,” in IEEE Transactions on Games, IEEE, 2021.

[43] K. D. Rogers, and A. A. Skabar, “A micromanagement task allocation system for real-time strategy games,” in IEEE Transactions on Computational Intelligence and AI in Games, vol. 6, no. 1, pp. 67-77, 2014.

[44] A. Heinermann, “Broodwar API,” https://github.com/bwapi/bwapi, 2013. [Online]. Available: https://github.com/bwapi/bwapi

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” preprint at https://arxiv.org/abs/1707.06347v2, 2017.

[46] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot et al., “Deep q-learning from demonstrations,” in Thirty-second AAAI conference on artificial intelligence, 2018.