University of Maribor

Faculty of Electrical Engineering and Computer Science

FERI

Time analysis of Machine Learning Algorithm utilization in complex Game Environments

Software Quality Analysis, Monitoring, Improvement, and Applications 2022 (SQAMIA 2022)

Authors:
Damijan Novak (damijan.novak@um.si), Domen Verber (domen.verber@um.si) and Iztok Fister Jr. (iztok.fister1@um.si)

12.9.2022

PRESENTATION AGENDA

1. Motivation and purpose of the article
2. Game domain
   - Game environment and their estimated game-tree complexities
   - Real-Time Strategy game environment
   - Games as the real-time systems
3. Machine Learning and Game Agents
   - Categories of Machine Learning
   - What are Game Agents?
4. Case studies
   - Case study one
     - Presentation of the case study one
     - Results and their interpretation
   - Case study two
     - Presentation of the case study two
     - Results and their interpretation
5. Future work

MOTIVATION

- Game environments and their connecting game spaces act as good simulation tools.
  - They are prosperous source of information.

- The Machine Learning (ML) algorithms can benefit from such simulation tools because a („never-ending“) input stream of data is suitable for fast and cheap testing.

- ML lessons learned in the game environments can hopefully be transferred to other research domains.

PURPOSE

- With this work, the ML algorithm utilization is made in the game domain.

- The time analysis is performed on two complex Real-Time Strategy game environment use cases.

- Considering the time component is crucial for understanding the ML time usages, allowing for even faster adoption of ML in all branches of software development.

GAME DOMAIN: GAME ENVIRONMENT AND THEIR ESTIMATED GAME-TREE COMPLEXITIES

- A game environment implements the rules of the game, game objects (along with all their properties pre-set by the developer), and connections (or relations) between these objects.

- Changing the objects‘ properties creates a new game variant (but it is still the same game!).

- Game space is defined as a high-dimensional set of unique game variants, with a point in game space being a specific vector of the game parameters.

GAME DOMAIN: GAME ENVIRONMENT AND THEIR ESTIMATED GAME-TREE COMPLEXITIES (2)

- Game spaces have large state spaces (i.e., the state space is defined as the set of all states that can occur in the game space), and action spaces (i.e., all possible actions that can be performed in the state space).

- The game tree presentation of a game space is better if the actual complexity of the game needs to be shown.

- In the game tree, the nodes are possible legal states of the game, and connections are possible legal actions that arise from the ground state and produce new states/nodes.

- Over such a game tree, the game-tree complexity can be given if the tree is searched in the „min-max style“ (i.e., an estimate of the number of positions that must be evaluated by searching with the min-max algorithm to determine the value of the initial state).

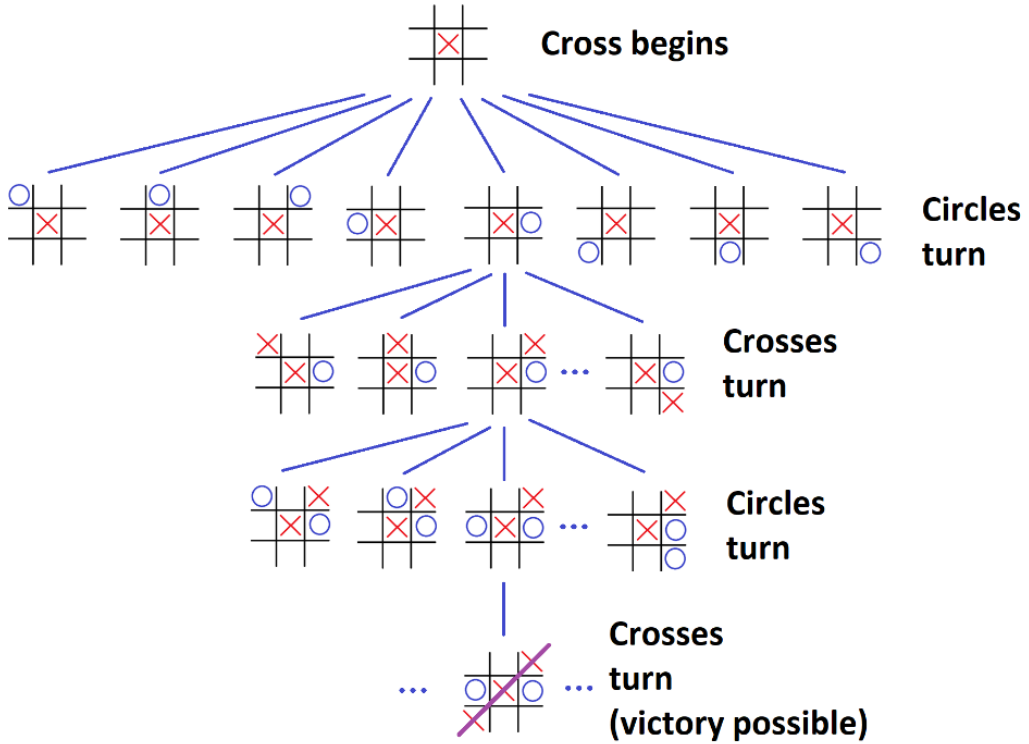
GAME DOMAIN: GAME ENVIRONMENT AND THEIR ESTIMATED GAME-TREE COMPLEXITIES (3)

- Game-tree complexity estimation is written as b^d, where b is the (estimated) average branching factor, and the symbol d is the (estimated) average depth of the game tree.

- Tic-Tac-Toe
  b ≈ 5 and d ≈ 9

- Chess
  b ≈ 35 and d ≈ 80

- Real-Time Strategy games
  b ≈ 30^{60} and d ≈ 36,000



GAME DOMAIN: REAL-TIME STRATEGY GAME ENVIRONMENT

- The adjective strategic in the name of Real-Time Strategy (RTS) games represents the highest form of the decision-making process the player performs while playing the game.
- In RTS games, players have to take into account gameplay aspects such as production and resource management, gameplay decisions (strategical command, tactical decisions, micromanagement of units, and the low-level unit behavior), scouting, map visibility, economy, and sometimes even diplomacy, to gain an edge advantage in the game.
- RTS games are challenging to master due to the high complexity of their game environments.

GAME DOMAIN: GAMES AS THE REAL-TIME SYSTEMS

- Real-time industry-standard DIN 44300 states:

„The operating mode of a computer system in which the programs for the processing of data arriving from the outside are permanently ready, so that their results will be available within predetermined periods of time; the arrival times of the data can be distributed randomly, or be already a priori determined depending on the different applications„.

- Games are almost always real-time systems because:
  - the game engines processes receive external data (e.g., from a user or an algorithm specified actions) on an ongoing basis,
  - the results are already reflected in the game state in the next game frame,
  - and the frames follow predetermined time intervals.
    - When a game runs with 30 Frames Per Second, approximately 33 ms (1,000 ms / 30) of the time slice is available with each frame.

MACHINE LEARNING

- ML is one of the branches of Artificial Intelligence.
- ML uses various statistical, probabilistic, and optimization techniques that allow the computer to learn from past experience (or examples) and detect difficult-to-observe patterns in large and complex data, which can also include noise.
- The ML field is immense, and it is divided into three categories:
  - Supervised learning: Examples of both input data and the associated results are provided. With these, the machine can learn to predict future output based on new (unseen) inputs.
  - Unsupervised learning: The outputs for the input data are not known in advance. Two main cases of utilization: a) The algorithm learns to classify data into categories not defined explicitly in advance, b) the automatic features extraction, which reduces the complexity of data representation.
  - Reinforcement learning: This does not require the dataset of the learning pairs. Its learning occurs based on a reward and a penalty while in direct contact with the environment (e.g., the game space).

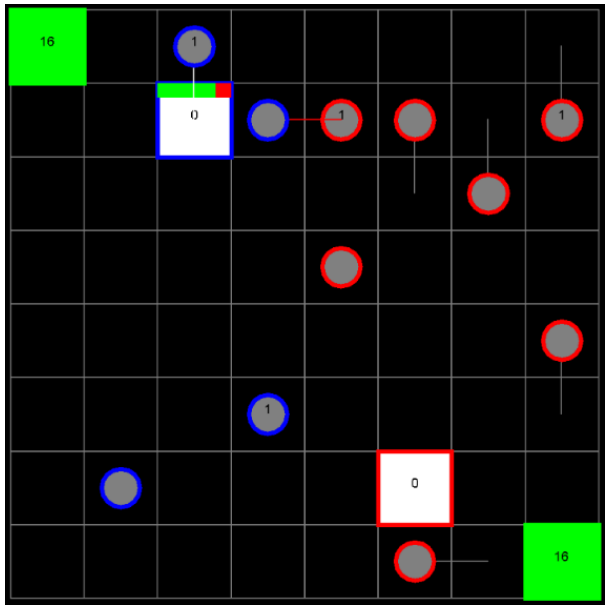
GAME AGENTS

- They are designed to play the game independently (i.e., without intermediate human intervention) and by following the game rules for which they were created strictly.
- The primary goal that Game Agents pursue is to try to win the game, and the secondary goal is to play the game in the best possible way.
- They are designed with various ML approaches, such as game trees (e.g., Monte Carlo Tree Search), Deep Learning (e.g., Deep Q Networks), Swarm Intelligence algorithms, etc.
- We used the following game agents (implemented by various researchers):
  - IDABCD: The Agent uses the classical alpha-beta (α-β) search technique, which was adapted specially for durative actions.
  - UCT: It is a standard and most popular Monte-Carlo Tree Search (MCTS) form, where an Upper Confidence Bound is used for Trees.
  - PuppetSearchMCTS: The Game Agent is designed using MCTS over a sequence of puppet moves (i.e., scripts with decision points exposed) and using α-β pruning.
  - NaiveMCTS: It is a standard form of the MCTS tree but with naïve sampling usage.
  - UCTUnitActions: It uses a standard UCT but is based on unit-actions (i.e., only the actions of one specific unit are considered in UCT nodes) instead of player-actions (i.e., considering all the actions coming from a player/agent).

CASE STUDIES

- The case study aims to acquire and analyze the time data when the ML-based Game Agents are involved in highly complex operations in the RTS game environment (in our case, the microRTS simulation environment).

- Two case studies were designed:
  - First, to capture the time data while the Game Agents act as playtesting agents. Game Agents were used for testing the validity of the game features (a general term used to describe what characterizes an individual game), which are part of the real-time game space.
  - Second, capturing the time data while the Game Agent plays the game and the ML methods are used to create its opponent model.



CASE STUDY ONE

- Seven game features were created for case study one, representing features usually used in the RTS game environments.
- Short descriptions of the game features:
  - GF1: is for the purpose of making the game state evaluations.
  - GF2: represents the game difficulty.
  - GF3: deals with the intermediate goal of the game, which is the structure construction.
  - GF4: utilizes the game engine and game objects.
  - GF5: belongs to the segment of in-game exploration.
  - GF6: is operating in a partially observable environment.
  - GF7: represents the game challenge to the player.
- Agents validated each game feature one hundred times (i.e., each agent played the game one hundred times during which the game feature was validated).
- The maximum time allowed for game feature validation was set to ten minutes (600 seconds) per game.

CASE STUDY ONE (2)

Table 1

Time intervals in seconds during all game feature validations for all the Game Agents used.

Game feature	IDABCD	UCT	PuppetSearchUCT	NaiveMCTS
GF1	[33.201, 265.664]	[12.477, 42.008]	[10.067, 21.705]	[12.151, 53.174]
GF2	[7.491, 174.423]	[10.567, 316.521]	[3.829, 201.98]	[21.931, 220.02]
GF3	[40.084, 183.178]	[21.055, 32.872]	[53.231, 56.572]	[21.307, 33.616]
GF4	[3.116, 86.894]	[2.699, 14.088]	[2.613, 17.444]	[2.538, 12.88]
GF5	[74.364, 242.159]	[21.669, 50.087]	[53.369, 111.86]	[19.784, 52.265]
GF6	[1.33, 3.97]	[1.436, 1.672]	[1.437, 1.743]	[1.474, 1.812]
GF7	[1.632, 6.709]	[1.826, 41.075]	[4.853, 5.223]	[1.776, 7.059]

CASE STUDY ONE (3)

Table 2

Average time values in seconds during all game feature validations for all the Game Agents used.

Game feature	IDABCD	UCT	PuppetSearchUCT	NaiveMCTS
GF1	101.704	21.769	14.396	19.517
GF2	32.280	127.676	30.147	67.516
GF3	111.487	27.408	54.435	25.782
GF4	25.8	6.608	6.752	5.915
GF5	127.001	38.046	74.743	33.858
GF6	1.565	1.528	1.567	1.617
GF7	3.049	4.359	5.046	3.098

CASE STUDY ONE (4)

Table 3

Standard Deviation values in seconds during all game feature validations for all the Game Agents used.

Game feature	IDABCD	UCT	PuppetSearchUCT	NaiveMCTS
GF1	52.931	5.267	2.324	6.594
GF2	20.194	65.636	29.183	33.361
GF3	29.681	2.311	0.768	2.942
GF4	17.778	2.168	2.457	2.161
GF5	30.844	5.902	11.609	5.651
GF6	0.53	0.05	0.054	0.062
GF7	0.879	5.732	0.067	1.41

CASE STUDY ONE (5)

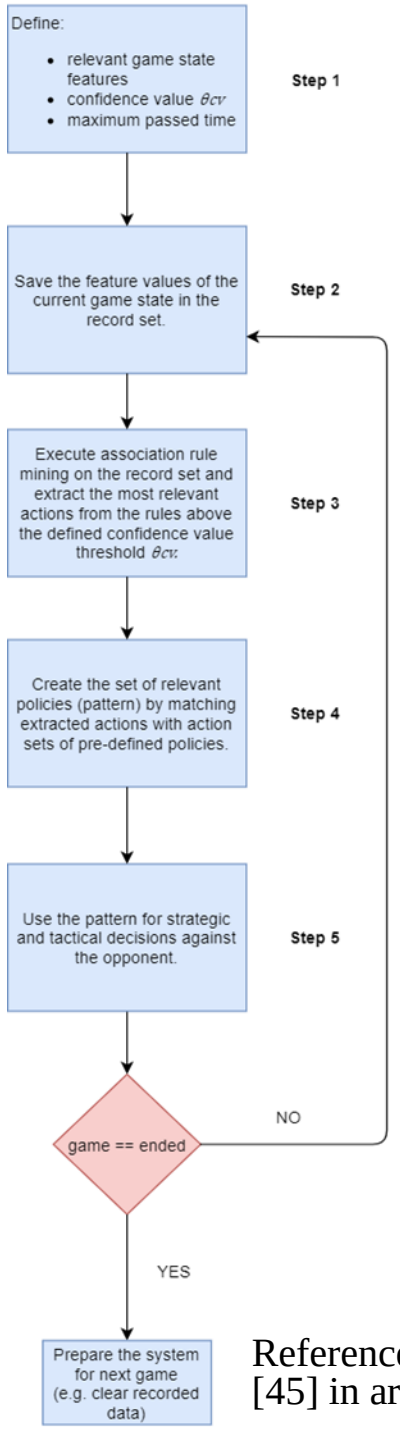
Main takeaways:

- The agent's time executions varied within the allowed upper execution time limit of 600 seconds due to the non-linear nature of ML algorithms. Still, it must be stated that all of them had completed game feature validation successfully inside that given limit.

- Case study one confirms that the game agents utilizing ML algorithms can deliver valuable results in the given time limit while also completing additional tasks (besides game-playing), such as validations.

CASE STUDY TWO

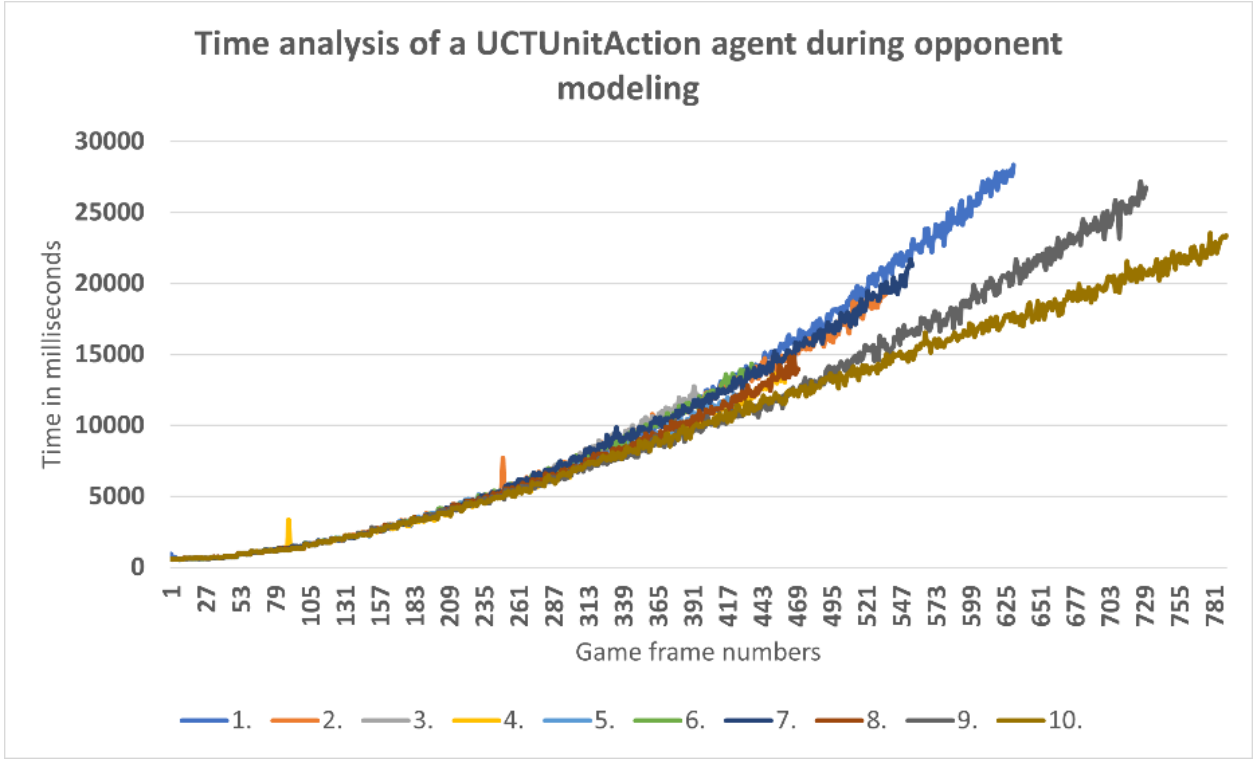
- For case study two, the time data were recorded while the UCTUnitAction agent was also playing the game, and its opponent model was created via incorporation of our ogpmARM method.
- The ogpmARM method utilizes Association Rule Mining (ARM), a valuable ML method for finding relationships between attributes in a transaction database.
- In our case, a variation of ARM called Numerical ARM (NARM) was used, which can work with categorical and numerical attributes.
- The NARM algorithms are based on stochastic population-based nature-inspired algorithms. The nature-inspired Differential Evolution algorithm was applied for case study two.
- Step three executes the NARM on the record set and extracts the most relevant actions from the NARM rules. The record set keeps all the feature values of the previous frames (i.e., the record set is not cleared after each frame). That way, the opponent modeling can take advantage of the full game information (from the first to the last frame of the game), and it is a good ML time analysis stress test, with constant expansion of the data that the NARM (and its DE algorithm) have to process.



Reference [45] in article.

CASE STUDY TWO (2)

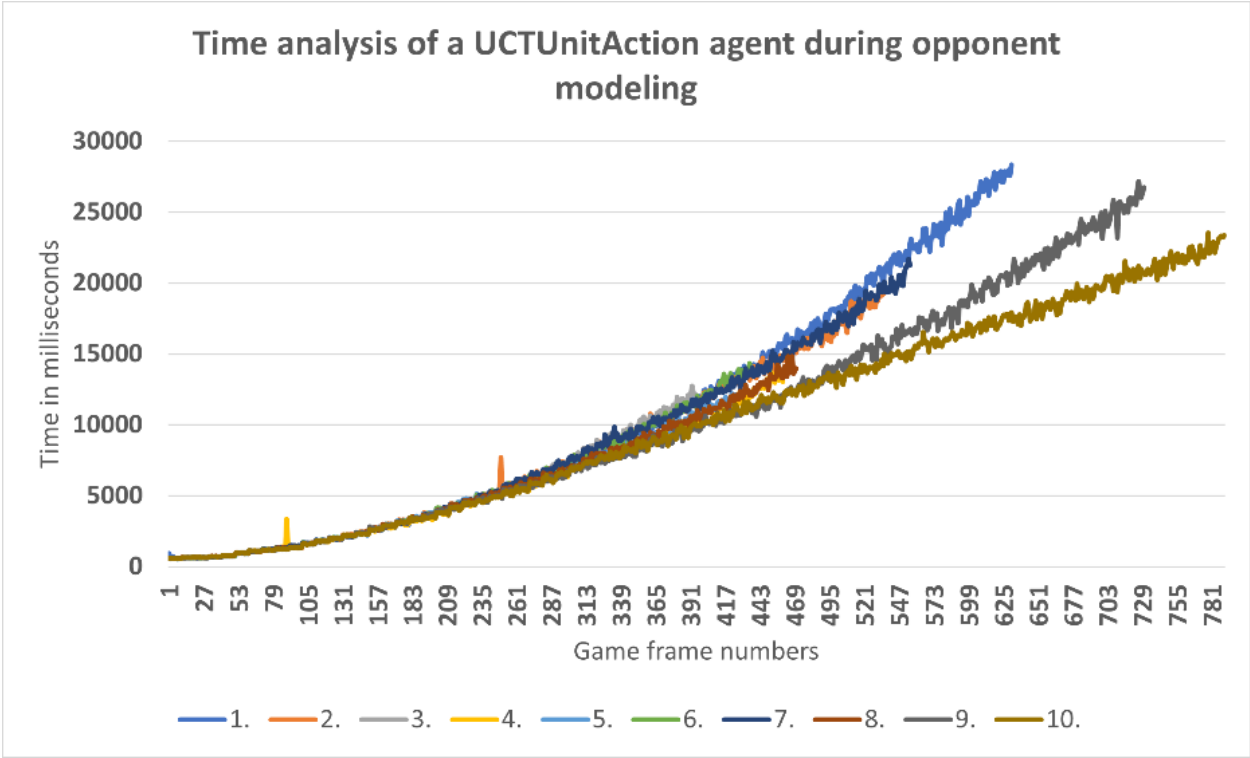
- The UCTUnitAction agents played the full microRTS game ten times on the standard 8x8 game map.
- Step three of an ogpmARM method was executed in each round and for each game frame.
- The game frame numbers are seen on the graph's abscissa axis, while, on the ordinate axis, the time is presented in milliseconds.



CASE STUDY TWO (3)

Main takeaway:

- From the beginning frame to around the 250th frame, all the games progress uniformly, and are all reaching the 5,000-millisecond mark. When progressing up to the 500th frame, the gameplay differences show variations of the measured time during each game.
- To the specific frame point, real-time data processing is possible because the time intervals are predictable, meaning that a real-time data limit could be set until which the operation is (certainly) completed.



CONCLUSIONS

- Both case studies gave valuable information regarding current ML usage possibilities while they are being utilized in such complex real-time cases.
- ML can deliver the results during the set time frame or comes very close to it (i.e., slight variations are possible due to the non-deterministic nature of ML algorithms). Case study one, therefore, showed that a ten-minute mark was enough to make one pass of the game while validating a game feature.
- In RTS games, the five-second processing interval would place it for possible in-game usage between reactive control of units (i.e., low-level controlling of units) and tactics usage (e.g., operating a group of units when in interaction with other (non-friendly) groups of units).
- If minor time variations are anticipated and soft real-time is allowed (i.e., the results are still valuable, even if they were acquired slightly over the limit), the utilization of ML in complex scenarios is certainly possible.

Thank you for your time. Please join the discussion.