Time analysis of Machine Learning Algorithm utilization in complex Game Environments

Damijan Novak, Domen Verber, Iztok Fister Jr.

Institute of Informatics, University of Maribor, Koroška Cesta 46, Maribor, 2000, Slovenia

Abstract

Nowadays, game environments are used not only for entertainment purposes such as playing, but also as simulation tools for various scientific researches. Machine Learning (ML) algorithms, on the other hand, are efficient Artificial Intelligence tools, used in software applications for purposes such as predicting data patterns or for software optimizations. In this work, a time analysis is performed on two use cases utilizing ML in a complex Real-Time Strategy game environment. The first use case deals with validation purposes of the real-time game space, and the second use case is designed for gathering time data during real-time ML usage. Considering the time component is crucial for understanding the ML time usages, and to allow for even faster adoption of ML in all branches of software development.

Keywords

Time analysis, game environment, Machine Learning, game use cases

1. Introduction

Nowadays, games are more than just an entertainment medium. They are also interesting for research, which otherwise doesn't need to connect directly to the expansion of knowledge in the Game domain. Domain areas such as Health [1], Cultural Heritage [2], Military studies and Education [3], Social studies [4], and many more, are all benefiting from them. That is because virtual game worlds are a prosperous source of information, and their data can serve as a (never ending) input stream to the algorithms/models/procedures being researched. For example, if researchers want to validate a novel algorithm, they could do that by running simulations within game worlds utilizing their diverse and rich game data. Not to mention that running such simulations is very cheap (e.g., it is cheaper to run simulations of drones flying in a packed virtual city than in the real world). The amount of data production within game worlds is also "infinite" (limited only by the computing power of modern computers), due to the interlinking of a large number of aspects (e.g., the aspect of the intertwining of environmental elements). The game world can be seen as a spatial form (i.e., spatial features) of the game, allowing the player to play a video game in a virtual world [5]. Also, note that the game world is connected closely to the game space (described in subchapter 2.1), because multiple game worlds can be created in one game space. For example, the game can be taking place in a virtual, as well as a physical world (e.g., trans-reality games [6]).

Machine Learning (ML), which is a sub-branch of Artificial Intelligence (AI), tries to use (past) experiences (e.g., experiences gained while operating in simulation environments), to improve the performance in interpreting patterns, based on which people or machines can then perform specific tasks [7]. Also, the concept of learning, in general, represents a model built on a sample of data (e.g., data classification), based on which people or machines make (future) decisions (e.g., on new/related data that were not previously used in building the model).

SQAMIA 2022: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 11--14, 2022, Novi Sad, Serbia

EMAIL: damijan.novak@um.si (D. N); domen.verber@um.si (D. V.); iztok.fister1@um.si (I. F.)

ORCID: 0000-0002-0834-3126 (D. N.); 0000-0003-1869-8286 (D. V.); 0000-0002-6418-1272 (I. F.)

In this work, the domain of ML is combined with the Game domain. Specifically, the ML is used in two complex use cases to gather time data, while the use cases are being executed. Such time data are necessary if we want to learn under what limits (i.e., time frames or time intervals) the usage of ML is possible. Specifically, with time analysis, we want to establish if the ML can be used in real-time (e.g., soft real-time).

The article is structured as follows. Chapter two presents the necessary domains related to the games and our work. The ML domain and game agents are explained in Chapter three. Chapter four presents both case studies. Conclusions are given in Chapter five.

2. Game domains and their related works

The presentation of the main game domains utilized in the use cases is made in this Chapter. The game environments and the theory behind estimating the time complexities when operating in such environments are described first. Second, the Real-Time Strategy (RTS) game environments are presented, along with the aspects that make them complex, and, therefore, capable simulation tools. Third, a description is provided regarding the real-time component of real-time games.

2.1. Game environments and their estimated time complexities

Game space can be described as an environment that 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. If the properties of the objects change, it can be said that it is still the same game, but in a different variation. In other words, each unique parameter setting of a specific game creates a new game variant, and the game space is defined as a high-dimensional set of unique game variants, with a point in game space therefore being a specific vector of the game parameters [8]. In modern times, game spaces are created mainly for commercial purposes (e.g., to sell a new mobile game), but they are also gaining traction for experimental evaluation [9] of algorithms, techniques, and models [10, 11]. Game spaces offer plenty of opportunities for tackling many research problems and possibilities for complex experiments, because the game spaces intertwine many aspects (e.g., hierarchial abstraction aspect used in decision process). That reflects both in the large state space and in the large action space. The state space is defined as the set of all states that can occur in the game space, and the action space is defined by all possible actions that can be performed in the state space. The state represents the set of game attributes and their values [12], and the action is the choice by which the player influences the state of the environment [13]. The set of all possible combinations of states and actions forms a full search space. But, such a full search space also includes the states that are illegal from the game rules point of view (e.g., a game board of the game Tic-Tac-Toe filled with only crosses). Therefore, when the game rules are applied (and the game always begins in the legal starting state), the size of the (legal) search space is reduced.

However, because the search space is probably not the best way to present the game in a manner by which the clarity of the actual complexity of the game would be shown, a game tree is preferred for such presentation. With such a tree, nodes are possible states of the game, and connections are possible legal actions that arise from the ground state and produce new states/nodes. The leaves of the search tree (or end nodes) are the states in which the game is decided. An example of a section of such a game tree for the Tic-Tac-Toe board game is shown in Figure 1.

The Figure 1 example demonstrates how the course of the game (i.e., individual states and the actions arising from them) can be presented with a game-tree. Over such a tree computational complexity can be given (game-tree complexity), which is presented as a tree search in the "min-max style" algorithm (i.e., it is 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), and is written with the estimated complexity of b^d [14]. The symbol b indicates the (estimated) average branching factor, and the symbol d is the (estimated) average depth of the game tree. The symbols b and d are often only estimates, and they can also be represented as a range interval, because the exact value cannot be given due to the magnitude of the problem. For a game of chess, such an estimate is $b \approx 35$ and $d \approx$

80, while, for one of the most complex game genres, the RTS games, the estimate is in the range of b ≈ 30^{60} and d ≈ 36,000 [15].

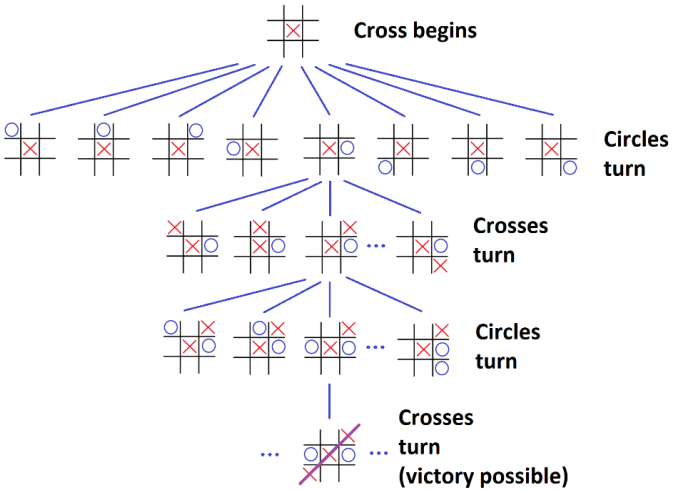

Figure 1: Game-tree representation for the Tic-Tac-Toe game

2.2. Real-Time Strategy game environments

RTS games are one of the subcategories of the computer Strategy game genre. The adjective strategic in the name of RTS games represents the highest form of the decision-making process the player performs while playing the game, as the player must make decisions constantly about the next moves (called actions). The notion of real-time, however, appears in the name because the game passes from one game state to another at such a speed (in the range of a few tens of milliseconds) that it seems to a person that the game takes place in real-time (more about the real-time component is explained in the next subchapter 2.3).

RTS games are most often created as (fantasy) combat simulations (e.g., the human race versus a fictional space race). Before the game starts every player has a choice regarding which side (or race) they want to play. Each side is shaped with specifics in the appearance, characteristics, and behavior of game defense units and buildings. Defense units are divided into mobile units (e.g., armored vehicles) and stationary units (e.g., gun cannons). In addition to gaming units, buildings (structures) also play an essential role. Buildings are used primarily to construct new defense units or other (more advanced) buildings. However, they can also perform particular functions, such as the production of the energy required for the operation of facilities, or they serve as fixed stationary points of defense (e.g., a wall). The group of all buildings in a limited space is, colloquially, called a base. The main building with which the game is started is often also named the base, and is of great importance because the player places other buildings around it. Loss of a base, however, can mean a big strategic loss, or even represent the end of the game.

RTS games are highly complex, as the player has to master and control a large number of aspects, such as [16]:

- Resource Management: An aspect focused on the collection/acquisition of virtual money or gaming raw materials such as wood, stone, etc.,
- Construction of new mobile gaming units and buildings: A player buys buildings with resources, which are often a necessary condition for the construction/purchase of mobile units,
- Micro and macro-management of units: In micro-management, individual units are managed in a shorter time period and on a smaller map scale (low-level behavior of units and smaller battles between units or groups of units), and in macro-management for a more extended period (development and construction of (more advanced) units, strategies, resource management for the successful construction of units supporting the current strategy at play, etc.),
- Tactical and strategic levels [17]: At the tactical level, the player develops a game plan for the range of thirty seconds, and at the strategic level, for around the span of three minutes into the future,

- Scouting: As the players don't always see the entire map, they must send one of the units to scout the map,
- Map visibility: Commercial games often have the option of only partial map visibility [18], meaning that the players can only see that part of the map they have explored and which is currently occupied by their units. The rest of the map is hidden under a so-called fog-of-war. However, for RTS research purposes, the entire state of the map is often visible (i.e., without concealing information).
- Economy: In scientific works, the individual aspects, such as resource management for the needs of the construction of mobile units and buildings, are often described under this term,
- Diplomacy: This aspect is not often used in RTS games, but it can take the form of grouping players into alliances, the disintegration of these alliances, sharing resources and helping each other, or using unwritten rules with which all the players agree.

Due to all these aspects, and with even more aspects possible (i.e., the list of aspects presented was non-exhaustive), it can be said that RTS game environments are one of the most complex game systems, due to the many moving parts which need to be synchronized to work in near real-time. Therefore, they are useful for scientific research and are excellent representatives of natural environments.

2.3. Games as the real-time systems

A system that can be said to operate in real-time must comply with the definition of the German industry Standard DIN 44300. The Standard states the following: "*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*" [19].

Games are real-time systems (or environments [20]), as they update the user iteratively with new information and graphic representation of the active environment on screen [21]. How the graphic information is displayed in games is very similar to the graphical expressions in movie technology. A movie is a sequence of frames which are slightly different from each other, so that they become a moving image when perceived by the human (brain) when displayed at a high enough speed (e.g., 24 Frames Per Second (FPS)). Games work similarly, but each game frame includes additional information such as the game state and the time slice available for taking actions (i.e., the time available before the actions will reflect in changes to the game state) between the game states transitions. Game engines are designed in a way as to always offer a time slice during the transition from one frame to another. During such transitions it is possible to obtain information about the game state (e.g., how many visible opponent units are on the map), as well as to influence this state via game engine methods (e.g., execution of actions for each friendly unit).

RTS games are often designed to operate at 24 [22] or 30 [23] FPSs, while, for other game genres, the number of FPS can reach 60 [24] or even more. So, when a game is running with 30 FPS, approximately 33 ms (1,000 ms / 30) of the time slice is available with each frame (note: The time slice can also be higher or lower, depending on the developer game engine settings), within which all chosen operations must be performed. If the game engine receives the actions (i.e., the operations that need to be executed) within the time slice, and the game engine is implemented correctly to guarantee all the action execution reliably (i.e., a hard real-time system which obeys the given time limits), then all the actions will be executed reliably during the transition from one frame to another. However, if we do not perform any operation (e.g., the calculations performed by the game agent exceed the time slice), the game engine won't wait, but will simply traverse to the next game frame.

Games are, therefore, almost always real-time systems, as they follow the definition written at the beginning of this subchapter. This conclusion is based on the facts that the game engine processes received 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 frame, and the frames follow predetermined time intervals. Also, the game engine can process the arriving data in real-time (i.e., when it receives

them) anytime during the available time slice, or group all the data and processes them all at once at the determined time. Note that, in both cases, the time slice must expire fully to the end, otherwise, the game display might experience artifacts.

3. Machine Learning and Game Agents

This chapter explains the main branches of ML and the game agents which build upon the ML methods.

3.1. Machine Learning

ML is one of the branches of AI, which uses a variety of statistical, probabilistic, and optimization techniques, that allow the computer to learn from past experience (or examples), and allow the detection of difficult-to-observe patterns in large and complex data, which can also include noise [25]. It is also the most effective method used in data analytics when a prediction about something must be made by devising models and algorithms [26].

The field of ML is immense. The ML algorithms can be divided into three main categories: supervised, unsupervised, and reinforcement learning [27].

With supervised learning we provide examples of bot input data and the associated results [28]. With these, the machine can learn to predict future output based on new (unseen) inputs. For outputs classes are often used, but the output can also be represented as a numeric value. An example of supervised learning would be classifying e-mails into spam and not-spam classes.

In unsupervised learning the outputs for the input data are not known in advance. There are two main cases in which unsupervised learning can be employed. In the first case, the algorithm learns to classify data into categories not defined explicitly in advance. From the previous example of e-mail classification, the algorithm must derive the spam/not-spam classes automatically. The second case is the automatic features extraction, which reduces the complexity of data representation. The algorithm must find a small number of data features of complex datasets that retain the original data's characteristics and essence [29].

Reinforcement Learning (RL) differs from supervised and unsupervised learning in that it 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). Actions are carried out iteratively within the environment, to accumulate and obtain the largest sum-total reward over a certain period [30].

The main tasks of machine learning can be summarized as follows: anomaly detection, association rule mining, clustering, binary and multiclass classification, forecasting, regression and object detection [31].

3.2. Game Agents

Game Agents 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 game's rules can be hardcoded-in by the developer, or acquired completely automatically (general gameplaying) [32]. Game Agents can: Utilize different Machine Learning methods (game trees [33], Deep Learning [34], Swarm Intelligence algorithms [35], etc.); operate in environments with different access to information (e.g., a partially observable environment [36]); have a different internal structure (e.g., modular design of the gaming agent [37]); and so on. 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 (e.g., to gain as many points in the game as possible) [38].

4. Case studies

The authors of different game domain works utilizing ML are sometimes also relating to the (real-) time components (e.g., exploring timings regarding the behavior switches in the RTS game StarCraft™ [16]). With this work, the initial time analysis case studies are conducted, with which the first steps are made towards the path of a more general time-domain picture of the game field. The case studies aim is to acquire and analyze the time data when the ML-based Game Agents are involved in highly complex operations in the RTS game environment. The analysis of such data should show if the usage of ML algorithms is possible in almost real-time (i.e., while the game is in progress, and, therefore, the ML algorithms aren't allowed to pre-process the game information beforehand). Two case studies were designed. First, to capture the time data while the Game Agents act as playtesting agents. Game Agents were used to test the validity of the game features (a general term used to describe what characterizes an individual game [39]), which are part of the real-time game space. Second, capturing the time data while the Game Agent plays the game, and while the ML methods are used to create its opponent model [40].

Both case studies ran on the following equipment:

- Hardware: Intel i7-9700 @ 3 GHz (turbo: 4.7 GHz) processor with eight cores, 32 GB RAM
- Software: Operating system Windows 10 Pro with the software development tools IntelliJ IDEA 2020.2.1 (Community Edition) in Java (version 13.0.2). The micro RTS's [41] (seen lower in Figure 2) latest version of the game simulation environment acquired from the authors' GitHub repository was used for both case studies.

4.1. Case study one

For the game space to be valid the correct implementation of the game features must be ensured (i.e., the game features must perform according to the design specifications during the game space planning phase). Any deviation from the intended operation is a bug, and can affect the course of the game, or, in the worst case, even cause the failure of the game. Therefore, it is in the game developer's interest to have game features with as few as possible bugs present, and that the testing is made in the most efficient way possible.

The solution to the problem is in the automation of validating the correctness of the game features by using customized game agents (ML-based) during the development of the game space. Four such game agents were utilized in case study one:

- IDABCD: The Agent uses the classical alpha-beta (α-β) search technique, and was adapted specially for actions which are durative [42]. The acronym ABCD therefore stands for Alpha-Beta Considering Duration. The ABCD is a modified and improved version (memory and time wise) of the minimax real-time game algorithm.
- UCT: It is the standard and most popular form of Monte-Carlo Tree Search (MCTS), where an Upper Confidence Bound is used for Trees [43]. The standard MCTS is otherwise a best-first search technique, and it is based on the stochastic simulations.
- 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 [44]. The framework is thus based on a search technique, which examines a problem that may arise during an adversarial search (i.e. planning into the future).
- NaiveMCTS: It is a standard form of the MCTS tree, but with the naïve sampling usage [41].

All the described agents are part of the existing microRTS package, and we used their default microRTS pre-set parameters (i.e., the parameters are set as the original authors decided them and weren't changed for the purpose of the experiment). Just to point out, it is worth mentioning that the PuppetSearchMCTS uses the default UnitTypeTable as input script, and the NaiveMCTS has

parameter values set to εl = 0,33, ε0 = 0,75 and max_depth = 1. The game agents had one hundred milliseconds time available for processing between frames, which is also the default value used in microRTS for experimentation purposes.

Seven game features were created for case study one, representing features usually used in the RTS game spaces. The game features are as follows:

- GF1: is for the purpose of making the game state evaluations. The evaluation result is calculated based on the current values of the game state (e.g., the number of combat units on the game map). It must be an accurate representation of the game state when the evaluation score is provided to the player. The microRTS map melee8x8Mixed6 was used, and the opponent was a RandomAI game agent (a basic agent executing random actions).
- GF2: represents the game difficulty. With this game feature, the opponent has an unfair advantage in the form of *x* number of units, which is reflected in the fact that the opponent defeats the player in each game. The microRTS map basesWorkers8x8 was used, and the opponent played by RandomBiasedAI (a basic agent executing random actions but with bias towards harvest, attack and return actions) had the help of five heavy units.
- GF3: deals with the intermediate goal of the game, which is the structure construction. Meaning that the game feature is designed to allow the players to build the structures (e.g., barracks) in the game. The microRTS map basesWorkers8x8 was used, and the opponent was PassiveAI (an agent which is not executing actions), to allow the game feature validating game agent to evolve without interruptions.
- GF4: utilizes the game engine and game objects. The player unit (e.g., combat unit) always hits with *x* points of damage. The microRTS map melee4x4light2 was used, and the opponent was RandomAI.
- GF5: belongs to the segment of in-game exploration (i.e., the player can unlock new technologies through the game technology tree usage). If the player chooses to explore the construction of specific units, they can build *x* player units (e.g., light combat units) using a specific building (e.g., through microRTS structure barracks). The microRTS map basesWorkers8x8 was used, and the opponent was RandomAI.
- GF6: is operating in a partially observable environment. The game feature states that the player cannot operate in an environment presenting partial information, and destroying enemy units hiding in such an environment is impossible. The microRTS map basesWorkers12x12 was used, and the opponent was RandomBiasedAI.
- GF7: represents the game challenge to the player. The player cannot destroy *x* buildings (e.g., barracks) that are guarded by *y* player units (e.g., a heavy combat unit) and are rushing the player, who has the z resource units available. The standard 8x8 microRTS game map was used, and the opponent, played by HeavyRush operating two barracks, was rushing the player with five heavy units.

Every agent 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. During the validation the time measurements were made, which consisted of time intervals showing the fastest and slowest validation time per game (Table 1), the average time (Table 2), and the Standard Deviation (Table 3) for all the games.

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]

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

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

For the time analysis of the agents the point of our focus is on the average execution times (Table 2), which, in our opinion, present the most precise time data picture. 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, but it must be stated that all of them, nonetheless, had completed game feature validation successfully inside that given limit. The IDABCD agent utilizes the allocated time slices where possible, based on the collected time data. That is to be expected, given that IDABCD builds on a (highly improved) version of the minimax real-time game algorithm, which is designed to use all the resources available to it. The UCT and NaiveMCTS agents stick to low execution times. The PuppetSearchMCTS agent is also expectedly fast, due to the use of scripts, which are keepers of game domain knowledge, which allows an agent to arrive at a decision quickly. Overall, 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.

4.2. Case study two

For case study two the time data were recorded while the UCTUnitAction agent (also part of the microRTS package) was also playing the game on the microRTS board (Figure 2), and its opponent model was created via incorporation of our ogpmARM method [45].

The UCTUnitActions agent 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). Such optimization avoids the exponential number of actions to some degree.

The ogpmARM method utilizes Association Rule Mining (ARM), a valuable ML method for finding relationships between attributes in a transaction database [46]. 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 [47] was applied for case study two.

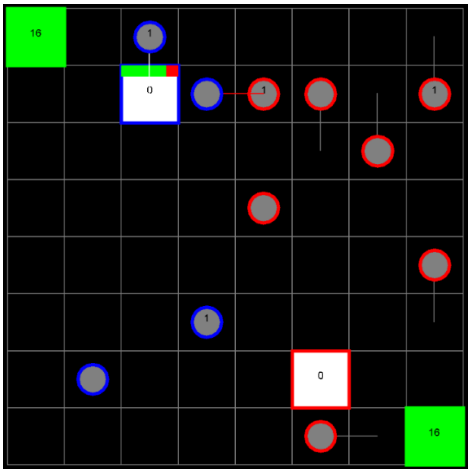

Figure 2: Game agent UCTUnitAction (represented with red units) during gameplay in the microRTS game environment

The method consists of five steps. The relevant game state features, confidence value and maximum time passed are defined in step one. Step two is used for saving the feature values in the record set (a set of feature values of the current game state). 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. Step four selects the policies (from the pool of pre-defined game policies) where the extracted actions are present, and forms a pattern. Step five uses the pattern for strategic and tactical decisions against the opponent.

For the case study two experiment the UCTUnitAction agents played the full microRTS game ten times on the standard 8x8 game map seen in Figure 2 against a RandomAI. Step three of an ogpmARM method was executed in each round and for each game frame.

The graph shown in Figure 3 shows the time data for all the ten games played. The game frame numbers are seen on the graph's abscissa axis, while, on the ordinate axis, the time is presented in milliseconds. 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 as variations of the measured time during each game. After the 500th frame, the time range of game five comes close to the 30 seconds mark, while game ten is still below the 25 seconds mark (even though the tenth game was longer based on the full frame count).

As the time analysis graph shows the functions of specific games are not linear, yet they are still predictable (i.e., a pattern is showing up to the some degree). For example, it could be said that, up to the 250th frame, the time functions are almost the same, while, up to the 500th frame even if they are not the same, they are still very close together (i.e., only slight variations between them). So, 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. Therefore, at least soft real-time limits are possible in complex environments such as RTS games while utilizing advanced ML algorithms.

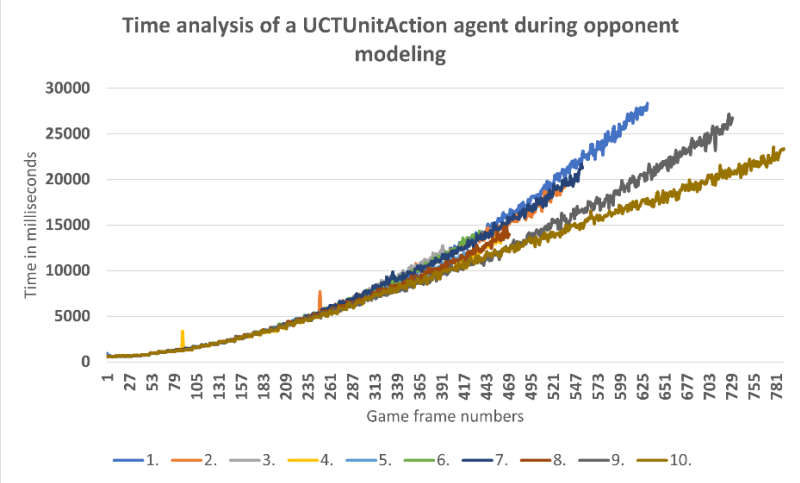

Figure 3: Time analysis of a UCTUnitAction agent during opponent modeling

5. Conclusions

In this article the time analysis was performed by utilizing different ML methods and algorithms used in complex RTS game environments. Two case studies were made for that purpose. Case study one dealt with the validation purposes of the real-time game space, and case study two was designed for gathering time data during opponent modeling of Game Agents with the help of the ML method. Both case studies gave valuable information regarding current ML usage possibilities while they are being utilized in such complex real-time cases. The data show that, if the time frames are set realistically (i.e., considering the problem at hand), the ML delivers the results during that 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. Case study two showed that the time predictions could be set in-line (i.e., scaled up) with the size of the problem, which is, in our case, the number of additional features that each frame adds. To put that into perspective, as Figure 3 shows, the time predictions are uniform to the 250th frame (the five-second) mark, which makes ML usage suitable for in-game use. For example, 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) [16]. But after that mark variations in time functions can occur, due to the complexities of the game environment and due to the non-determinism of ML methods, but even then, the time limits needed for ML processing are predictable for many more frames ahead. Therefore, if such 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.

In the future, we envision that even more ML usage will occur in all software process segments, either while the software is being developed or is in production use. So, our future plan involves additional experiments delving deeper into the time components, such as the variations seen in the case two study.

6. Acknowledgments

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

References

[1] M. A. Ahmad, D. K. A. Singh, N. A. Mohd Nordin, K. Hooi Nee, and N. Ibrahim. "Virtual reality games as an adjunct in improving upper limb function and general health among stroke survivors." International journal of environmental research and public health (2019), 16(24): 5144.

[2] M. Ćosović, and B. R. Brkić. "Game-based learning in museums—cultural heritage applications.", Information (2019), 11(1): 22.

[3] M. Macedonia. "Games, simulation, and the military education dilemma." In Internet and the University: 2001 Forum (2002), Cambridge: MA: Educause, 157-167.

[4] B. M. Maguth, J. S. List, and M. Wunderle. "Teaching Social Studies with Video Games." The Social Studies (2015), 106(1): 32-36, DOI: 10.1080/00377996.2014.961996

[5] O. T. Leino. "From game spaces to playable worlds." In The Philosophy of Computer Games Conference (2013), Bergen, University of Bergen, 2-4.

[6] C. A. Lindley. "Game space design foundations for trans-reality games." In Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology (2005), 397-404.

[7] B. Geisler. "Integrated machine learning for behavior modeling in video games." In Challenges in game artificial intelligence: papers from the 2004 AAAI workshop. AAAI Press (2004), Menlo Park, 54-62.

[8] A. Isaksen, D. Gopstein, and A. Nealen. "Exploring Game Space Using Survival Analysis." In FDG (2015).

[9] M. Buro, and T. Furtak. "RTS games as test-bed for real-time AI research." In Proceedings of the 7th Joint Conference on Information Science (JCIS) (2003), volume 2003: 481-484.

[10] K. Shafi, and H. A. Abbass. "A survey of learning classifier systems in games." IEEE Computational intelligence magazine (2017), 12(1): 42-55.

[11] G. N. Yannakakis, and J. Togelius. "A panorama of artificial and computational intelligence in games." IEEE Transactions on Computational Intelligence and AI in Games (2014), 7(4): 317-335.

[12] C. J. Harrington, U.S. Patent No. 10,220,313, Washington, DC: U.S. Patent and Trademark Office, Filed March 10th., 2017, Issued March 5th., 2019.

[13] Z. J. Pang, R. Z. Liu, Z. Y. Meng, Y. Zhang, Y. Yu, and T. Lu. "On reinforcement learning for full-length game of starcraft." In Proceedings of the AAAI Conference on Artificial Intelligence (2019), 33(1): 4691-4698.

[14] G. Synnaeve. "Bayesian programming and learning for multi-player video games." Ph.D. thesis, Universite de Grenoble, 2012.

[15] G. Synnaeve, and P. Bessière. "Multiscale Bayesian modeling for RTS games: An application to StarCraft AI." IEEE Transactions on Computational intelligence and AI in Games (2015), 8(4): 338-350.

[16] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. "A survey of real-time strategy game AI research and competition in StarCraft." IEEE Transactions on Computational Intelligence and AI in games (2013), 5(4): 293-311.

[17] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. "RTS AI Problems and Techniques." (2015), Springer International Publishing, Cham, 1-12.

[18] V. Zammitto. "Visualization techniques in video games." Electronic Visualization and the Arts (EVA) (2008), 267-276.

[19] W. A. Halang, R. Gumzej, M. Colnarič, and M. Družovec. "Measuring the performance of real-time systems." Real-time systems (2000), 18(1): 59-68. doi:10.1023/A:1008102611034.

[20] P. I. Cowling, M. Buro, M. Bida, A. Botea, B. Bouzy, M. V. Butz, P. Hingston, H. Muñoz-Avila, D. Nau, and M. Sipper. "Search in real-time video games." Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2013).

[21] S. Kaskela. "Applying real-time user interface practices to game." (2019)

[22] 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).
[23] T. Hall, and M. Magnusson. "Adaptive Goal Oriented Action Planning for RTS Games." (2010)
[24] D. Andreev. "Real-time frame rate up-conversion for video games: or how to get from 30 to 60 fps for "free"." In ACM SIGGRAPH 2010 Talks (2010), 1-1.
[25] J. A. Cruz, and D. S. Wishart. "Applications of machine learning in cancer prediction and prognosis." Cancer informatics 2 (2006): 117693510600200030.
[26] S. Angra, and S. Ahuja. "Machine learning and its applications: A review." In 2017 ICBDAC, IEEE (2017), 57-60.
[27] S. R. Salkuti. "A survey of big data and machine learning." International Journal of Electrical & Computer Engineering (2020), 10(1): 575-580.
[28] J. Sethi, and M. Mamta. "Ambient air quality estimation using supervised learning techniques." EAI Endorsed Transactions on Scalable Information Systems (2019), 6(22).
[29] L. Wang. "Discovering phase transitions with unsupervised learning." Physical Review B (2016), 94(19).
[30] L. Galway, D. Charles, and M. Black. "Machine learning in digital games: a survey." Artificial Intelligence Review (2008), 29(2): 123-161.
[31] S. Russell, P. Norvig. "Artificial Intelligence: A Modern Approach.", 4th edition, Pearson, 2022.
[32] G. Kuhlmann, and P. Stone. "Automatic heuristic construction in a complete general game player." AAAI (2006), volume 6.
[33] R. K. Balla, and A. Fern. "UCT for tactical assault planning in real-time strategy games." In Twenty-First International Joint Conference on Artificial Intelligence (2009).
[34] P. A. Andersen, M. Goodwin, and O. C. Granmo. "Deep RTS: a game environment for deep reinforcement learning in real-time strategy games." In 2018 IEEE conference on computational intelligence and games (CIG) (2018), IEEE, 1-8.
[35] A. R. Tavares, G. L. Zuin, H. Azpúrua, and L. Chaimowicz. "Combining genetic algorithm and swarm intelligence for task allocation in a real time strategy game." Journal on Interactive Systems (2017), 8(1).
[36] A. Ouessai, M. Salem, and A. M. Mora. "Online Adversarial Planning in μRTS: A Survey." In 2019 International Conference on Theoretical and Applicative Aspects of Computer Science (ICTAACS) (2019), volume 1, 1-8.
[37] C. Si, Y. Pisan, and C. T. Tan. "Automated terrain analysis in real-time strategy games." In FDG (2014).
[38] T. Joppen, T. Strübig, and J. Fürnkranz. "Ordinal bucketing for game trees using dynamic quantile approximation." In 2019 IEEE CoG (2019), IEEE, 1-8.
[39] S. Heintz, and E. L. C. Law. "Digital educational games: methodologies for evaluating the impact of game type." ACM Transactions on Computer-Human Interaction (TOCHI) (2018), 25(2): 1-47.
[40] F. Schadd, S. Bakkes, and P. Spronck. "Opponent Modeling in Real-Time Strategy Games." In GAMEON (2007), 61-70.
[41] S. Ontañón. "The Combinatorial Multi-Armed Bandit Problem and its Application to Real-Time Strategy Games." In AIIDE (2013), 58 - 64.
[42] D. Churchill, A. Saffidine, and M. Buro. "Fast heuristic search for RTS game combat scenarios." In Eighth Artificial Intelligence and Interactive Digital Entertainment Conference (2012).
[43] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. "A survey of monte carlo tree search methods." IEEE Transactions on Computational Intelligence and AI in games (2012), 4(1): 1-43.
[44] N. A. Barriga, M. Stanescu, and M. Buro. "Game tree search based on nondeterministic action scripts in real-time strategy games." IEEE Transactions on Games (2017), 10(1): 69–77.

[45] D. Novak, and I. Fister Jr. "Adaptive Online Opponent Game Policy Modeling with Association Rule Mining." In 2021 IEEE 21st International Symposium on Computational Intelligence and Informatics (CINTI) (2021), IEEE, 259-266.

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

[47] I. Fister Jr., A. Iglesias, A. Galvez, J. D. Ser, E. Osaba, and I. Fister. "Differential evolution for association rule mining using categorical and numerical attributes." In International conference on intelligent data engineering and automated learning (2018), Springer, Cham, 79-88.