University of Maribor

Faculty of Electrical Engineering
and Computer Science

FERI

Turn Action Extractions into Game Agent Module Decisions

32th International Electrotechnical and Computer Science Conference ERK 2023

Authors:
Damijan Novak (damijan.novak@um.si) and Iztok Fister Jr. (iztok.fister1@um.si)

28.9.2023

Presentation agenda

Motivation

- Traditional approaches in commercial games:
  - Game agents rely on hard-coded conditional statements (scripted agents), while others employ structured internal logic, such as decision trees, behavior trees (similar to decision trees), finite state machines, etc.
  - Games are typically designed to ensure that game outcomes are fully predictable (deterministic).
- Our motivation is to integrate a Machine Learning (ML) repertoire of tools into existing modules (i.e., modularly designed game agents) to enhance their functionality and make them more adaptive.

Module = practice of breaking down problems into smaller, distinct challenges.

Purpose

We propose two primary contributions:

1. Our first contribution is the introduction of the Self-Preservation Module (SPM). This module is designed to instill a sense of self-preservation in game units.

2. Our second contribution focuses on enhancing module decisions with online adaptivity. This is achieved by incorporating opponent action extractions through Numerical Association Rule Mining (NARM). This process leverages advanced ML optimization algorithms, such as Differential Evolution.

Real-Time Strategy games

- Real-Time Strategy (RTS) games have emerged as one of the premier testbeds for Artificial Intelligence (AI) research within the game domain.
- RTS games present many research challenges: RTS games are complex environments, often characterized by only partial information availability.
- To successfully defeat an opponent, players must navigate various domains of expertise, including tactical and strategic planning.
- Game agents have the potential to emulate human-like attributes such as creativity, adaptability, the element of surprise, varying activity levels, strategic foresight, intuition, and perhaps most fundamentally, for all sentient beings, the inherent will to survive.

Offline vs. Online adaptivity

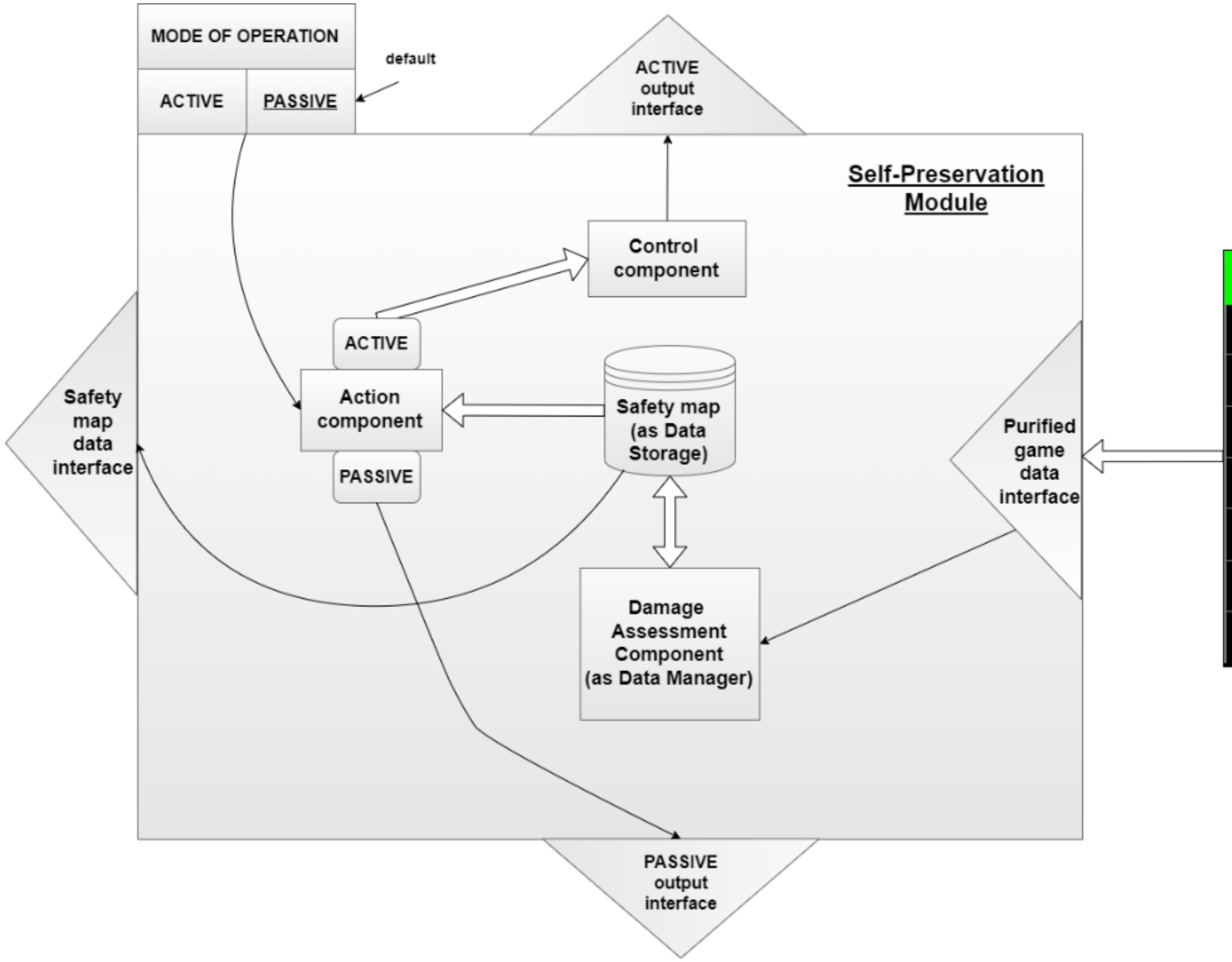
Offline adaptivity:

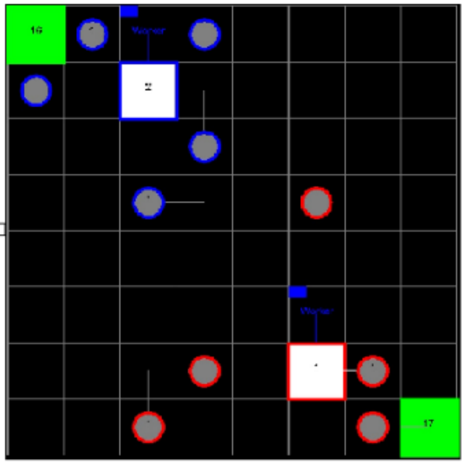
- ❖ Game data, such as relevant in-game observations (opponent actions and the like), is collected during gameplay. However, the assessment of this information occurs only after the game has concluded.
- ❖ The data can either be processed after the completion of a game or the next time the game environment is loaded.
- ❖ Offline adaptivity is typically the preferred method for processes that demand significant computational resources. This includes handling large volumes of data or employing computationally intensive algorithms, which might be impractical to run during live gameplay.

Online adaptivity:

- ❖ This approach is suited for algorithms capable of almost real-time processing, converting input into output swiftly.
- ❖ Appropriate in-game actions can be determined and executed on time, ensuring their relevance and impact.

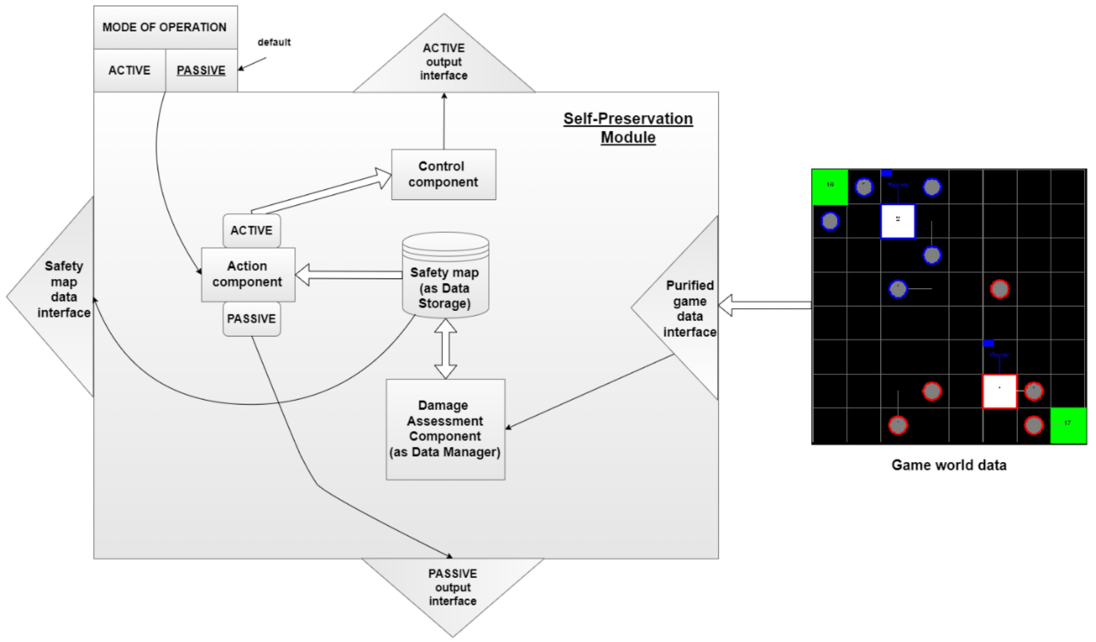
Self-Preservation Module Architecture




Game world data

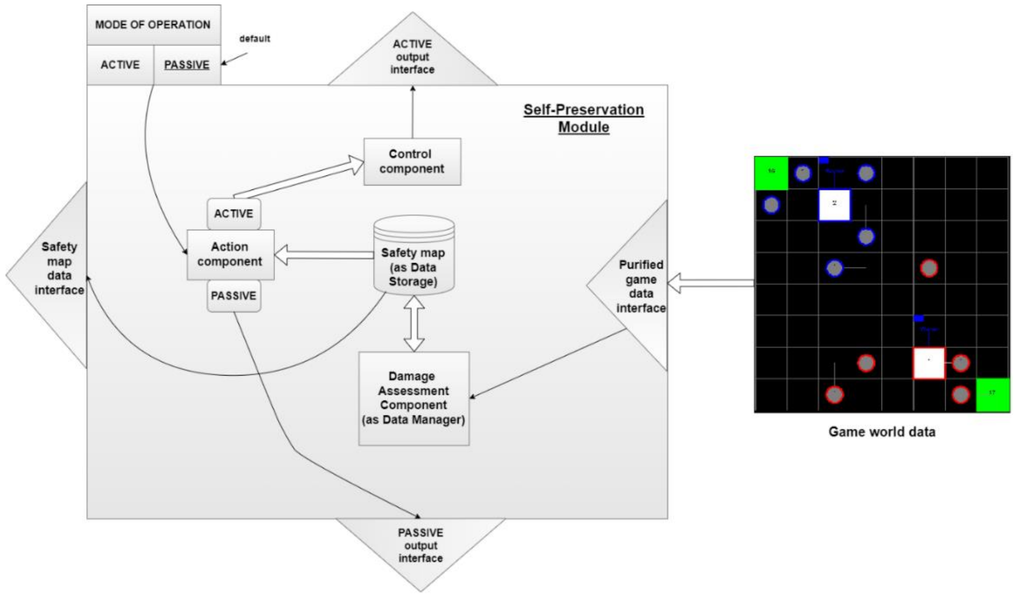
Self-Preservation Module Architecture (2)

- The **Self-Preservation Module** (SPM) is designed to instill a sense of self-preservation in game units.
- Purpose of the SPM system:
  - Analyze the current game state.
  - Make tactical/strategic decisions to ensure unit safety.
  - Emulate the human instinct for self-preservation.
  - Specifically crafted to complement, not interfere with, existing components.
  - Enable non-invasive decisions, granting control over the self-preservation behavior of all units.
- Roles within SPM:
  - Passive
  - Active
- The central component of the SPM is the Safety map.
- All auxiliary components, whether directly or indirectly, connect to the Safety Map.
- The Safety map = a purified version of the current game world state and relevant historical data about past events. This combined information is vital to assess threats and ensure the units' safety.



Self-Preservation Module architecture (3)

- The Self-Preservation Module (SPM) encompasses three additional components: Damage Assessment Component (DAC), Action Component (AC), and Control Component (CC).
- Notably: The Control Component (CC) becomes operational solely when the SPM's mode of operation is designated as 'active'.
- SPM communicates with other modules through four well-defined interfaces: Purified Game Data Interface (PGDI), Passive Output Interface (POI), Active Output Interface (AOI), and a Safety Map Data Interface (SMDI).
- By default, the SPM's update method is initiated between every frame, corresponding to the time slots when the game agent processes its moves.
- However, depending on requirements, this initiation can be delayed, perhaps every second, third frame, and so on.
- Crucially, the update method solely requires the current game state for its operation.



Damage Assessment Component (DAC)

For each SPM update:

1. PGDI first extracts relevant feature information (e.g., damage done to friendly units in the previous frame) and passes it to the DAC.
2. Subsequently, DAC updates the state of the Safety map.

The DAC updates the Safety Map by considering many factors (e.g., a unit has sustained damage). Notably, the action extraction data, as highlighted by the blue-colored lines from 17 to 19 in the DAC algorithm, indicates instances where changes in opponent behavior are utilized to refresh the tree.

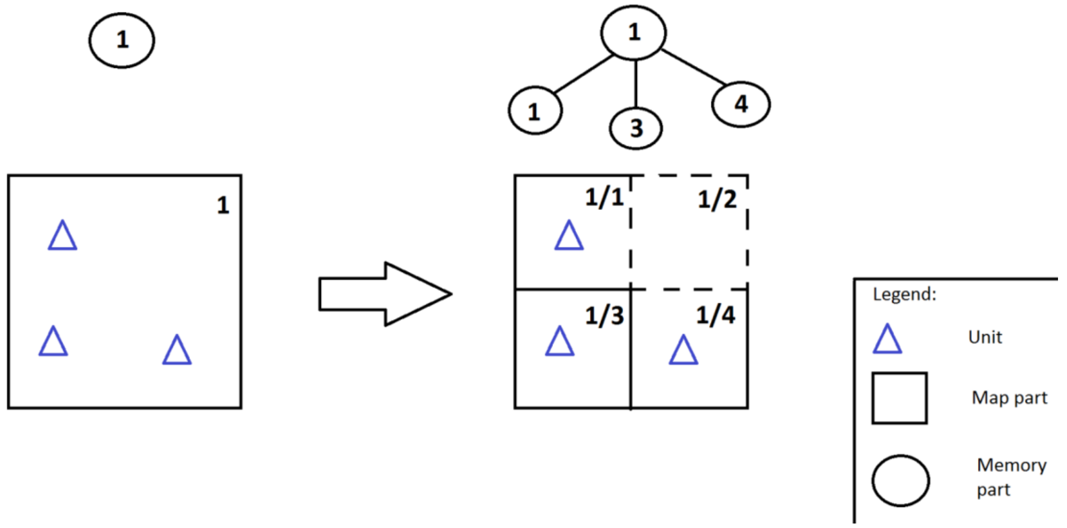
Algorithm 1: Pseudo-code of DAC update procedure of the Safety map

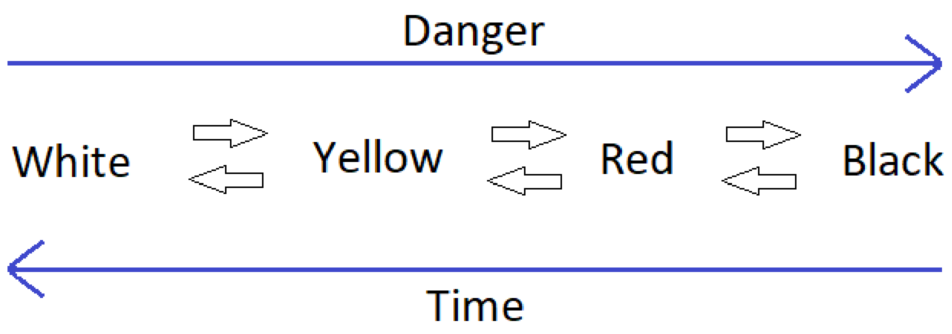
```
     // Initial parameter settings:
     // - color and state parameter at which point we divide cells
     // - color and state parameter at which point we destroy cells
     // - unit destroyed parameter
     // - unit was hit parameter
     // - opponent behavior changed parameter
     // - decremental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

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

Safety map

- The Safety Map, as a data storage system, is dynamically designed, resembling the structure of a quad tree (a tree data structure that can have up to four children).
- The color of each cell changes constantly (color threat levels are introduced).





Action extraction driven module choices: Action extraction process

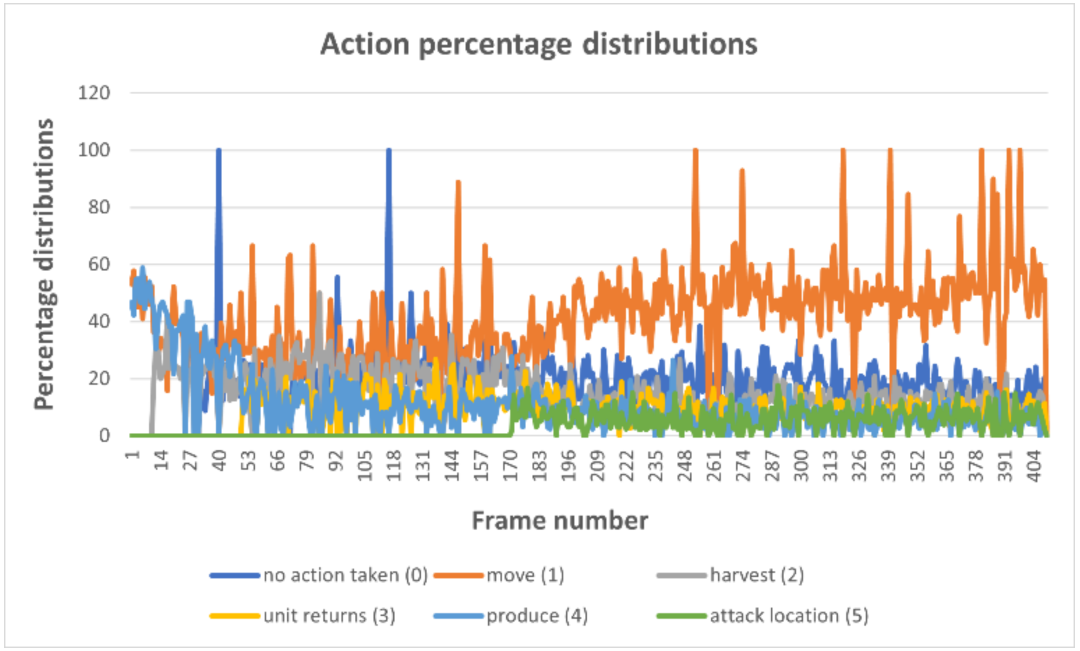
- ❖ Association Rule Mining (ARM) = a data mining technique that finds data patterns representing relationships between items.
- ❖ To effectively extract relevant information, the Numerical Association Rule Mining (NARM) method was used.
- ❖ While ARM primarily deals with binary data, NARM is more versatile and can also handle numerical data.
- ❖ The association rules were derived based on the gaming state and actions recorded from the beginning of the game.
- ❖ Only the association rules that contain the opponent's action and high-enough confidence were kept.
- ❖ Each action's frequency (the number of occurrences) is calculated.
- ❖ The result is a support vector of each possible (valid) opponent's actions.

Experiment

- The experiment was designed in microRTS simulation environment, with a purpose of providing the live game data for the NARM action extraction process.
- Features: The number of friendly / enemy workers / light units / heavy units / ranged units, a flag if a friendly base has been threatened, friendly / enemy resources left, number of friendly / enemy bases, and the number of enemy barracks.
- Actions: No action taken (0), move (1), harvest (2), unit returns (3), produce (4), and attack location (5).
- The opponent chosen for the study: UCT (Upper Confidence Bounds applied to game tree), which played against the basic built-in RandomAI.
- The data used for NARM were gathered from the game states across the whole game.
- In each game state, the processing set holds the feature values and opponent actions of the current and all the game states before it.
- The threshold for the rules from which the actions are extracted is set to 0.5.

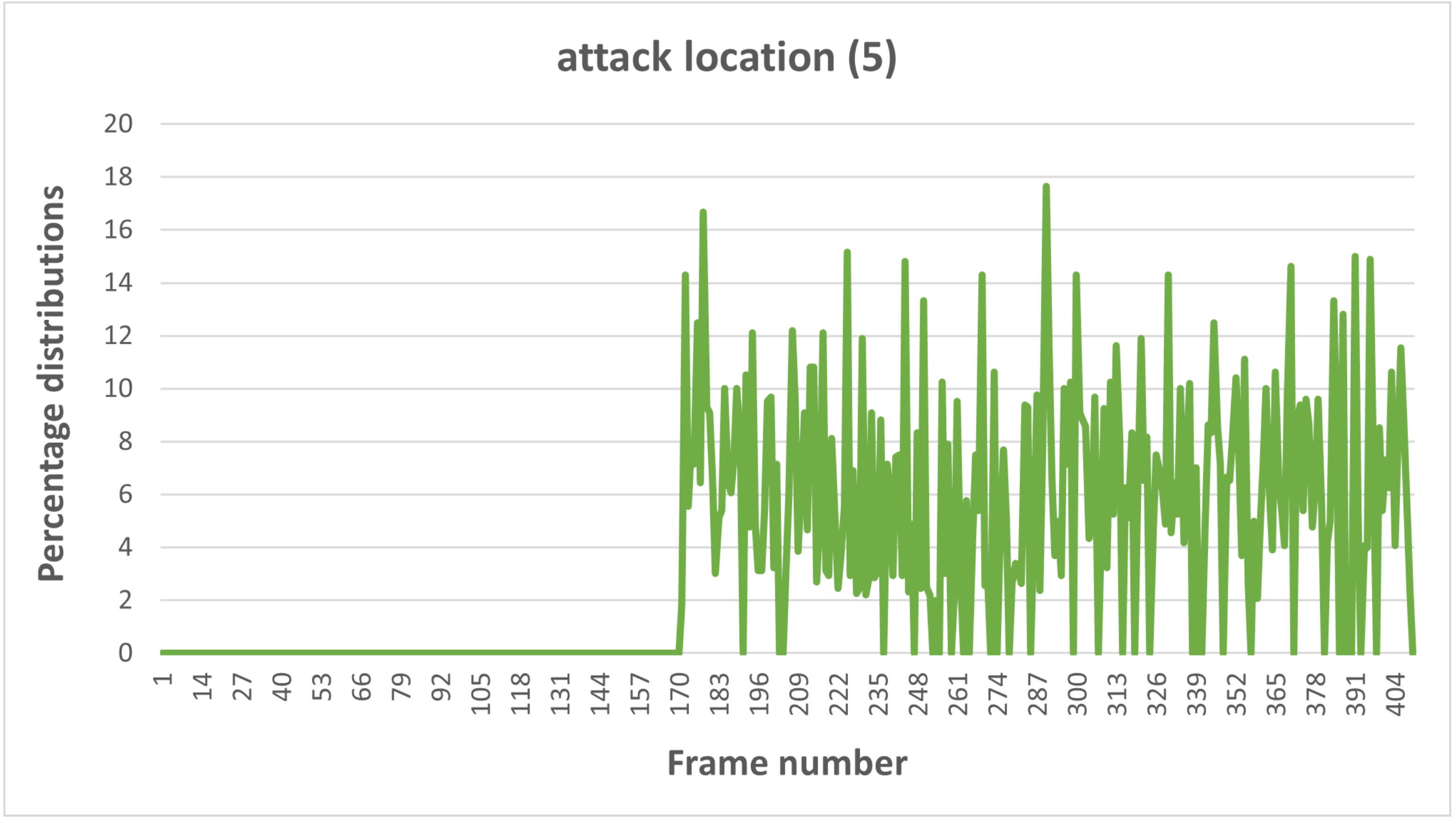
Results

- The graph of the recorded data of a UCT agent during gameplay throughout the whole game.
- The abscissa axis indicates the consecutive frame numbers, while the ordinate axis shows the percentage distributions of each action for that specific frame.
- SPM responds to an opponent's action behavior. It's worth noting that some of these moves can significantly influence the game's outcome.



Results (2)

Only the percentage data of the action attack location (5) was isolated, to show better how the UCT agents' attack distribution changes with passing game time.



Final thoughts

- The module is universal, and can be included in every game agent using a modular design.
- During initial testing it was noticed that, at the beginning of gameplay, the action extraction processing was possible in online mode, while, later on, when the sets of game features and actions stacked up, the game frame time slices (100 milliseconds) were exceeded, and the module operation resembled more that of the offline mode.
- The data gathered show encouraging results for use in module decision-making.
- The opponent data show an evident specter of action variations` (i.e., ups and downs seen on the two graphs representing the agents' behavior changes) usages across the whole game span. Overall, such distinct opponent action variations are crucial for module gameplay decision-making.

Thank you for your time. Please join the discussion.