# Building a Trace-Based System for Real-Time Strategy Game Traces

Stefan Wender[1], Amélie Cordier[2], and Ian Watson[1]

[1] The University of Auckland, Auckland, New Zealand
[2] Université Lyon 1, LIRIS, UMR5205, F-69622, France
s.wender@cs.auckland.ac.nz, amelie.cordier@liris.cnrs.fr, ian@cs.auckland.ac.nz

**Abstract.** We describe the conception of a visualization and transformation tool for traces of the real-time strategy (RTS) computer game StarCraft. The development of our tool StarTrace is driven by the domain the traces originate from as well as the observable elements those traces contain. We elaborate on those influences, which also include both the structure of the existing game traces and the requirement to use these traces to improve the performance of an machine learning (ML) agent that attempts to learn to play parts of the game. We then describe the architecture of the browser-based tool and the trace model behind it. The purpose of StarTrace is to eventually improve the learning process of this agent by providing the means to harness the enormous amount of data included in complex RTS games. Finally, an example application showcases how the tool can help to better understand the player behavior stored in game traces.

## 1 Introduction

RTS games, such as StarCraft, provide a challenging test bed for AI research. They offer a polished environment that includes numerous properties such as: incomplete information, spatial and temporal reasoning as well as learning and opponent modeling that are interesting for AI research [1]. For this reason we chose StarCraft as a testbed for a machine learning (ML) approach that tries to learn how to manage combat units on a tactical level ("micromanagement")[2]. The tool described in this paper is part of this effort to create an autonomous agent that uses ML techniques such as reinforcement learning (RL), CBR and Trace-based reasoning (TBR) to solve the complex problem of micromanaging units in StarCraft. StarTrace is conceived to facilitate the use of StarCraft traces by making the information contained in those traces more accessible and understandable.

The quality and complexity of StarCraft has made it very popular, which in turn has led to a vast body of StarCraft traces, so-called "replays", that are readily available online. These replays contain implicit expert knowledge in the form of recorded actions of the players, basically traces of their gameplay. TBR [3] is a paradigm that helps us to make this knowledge explicit and reusable by agents.

There are several incentives for applying TBR to the problem of managing combat units in StarCraft. The large amount of expert knowledge that is available online comes in the form of game replays which have an inherently trace-like structure. This expert knowledge can be extracted from the replays and, with the right tools and transformations, used to improve the control of combat units by the agent.

Furthermore traces can improve the general learning process of an agent trying to learn how to perform tasks inside the game. Currently our agent uses a hybrid ML approach that involves case-based reasoning (CBR) to manage game information. The case representation is based on the environment state and unit attributes. The use of traces of unit attributes instead of attributes from a single point in time can add valuable information and thus improve the learning process in a number of ways [4].

In this paper we describe the conception and development of our tool StarTrace, a browser-based application to visualize and transform traces of RTS game data. The ability to create trace transformations is one of the main features of the tool. Its graphical interface allows easy specification and modification trace transformation criteria. The data that is displayed and modified by StarTrace is either obtained from previously recorded games or by actively monitoring user interactions during gameplay.

## 2 StarTrace

### 2.1 Hybrid CBR/TBR and Reinforcement Learning for Micromanagement in RTS Games

The aim in an RTS game such as StarCraft is to manage an economy by collecting resources and buildings, to create combat units and to eventually eliminate all enemies with the help of those units. Choosing and managing a strategy is one of the major AI research problem areas in RTS games. Micromanaging units at a tactical level is another, equally important area.

Presently our agent uses a hybrid CBR/RL approach to learn how to manage those units in combat situations. Cases are based on single units and consist of a case description, the current state of the game environment, as well as a solution, which is a set of all possible actions for the currently active unit. Each action has a value assigned to itself that represents its fitness. Since the number of possible states the game environment can be in is huge, abstraction is needed for the representation. The environment state is abstracted into influence maps to represent the areas of influence for opposing and own units. If we regard these influence maps as simple greyscale pictures with intensity values representing the influence values, the pictures can in turn be converted into histograms to make them usable as an efficient similarity metric. A nearest neighbor (NN) retrieval of cases in the case base is done based on the histograms and furthermore based on other attributes specific to the single unit in question.

### 2.2 Trace Recording

The extraction of primary, unaltered traces from StarCraft can happen in two ways. On the one hand, game traces can come from game replays, i.e. recorded games that have already been played. StarCraft provides functionality to record games in a standardized way that produces such primary game traces with a common model (M-Traces) regardless of where and when games are recorded.

On the other hand, traces can also be recorded from active gameplay to create M-Traces similar to those created by the game itself. This behavior is comparable to other approaches that build trace-based systems which record user interactions [5]. As a result both actively recorded M-Traces and those stored as replays have an identical structure irregardless of provenance.

Replays are simply a recording of all actions a player performs during a game, inherently traces of user interactions with the game. Figure 1 shows the structure of such a trace. Expert players can perform several hundred of those recorded user interactions per minute. As can be seen in the structure of the replay, the type of actions can vary greatly and involves differing numbers of parameters.

| Time | Player | Action | Parameters | Units ID |
|---|---|---|---|---|
| 17728 | Ryan[Shield] | Move | (1455,1804),0,228,0 | 0 |
| 17728 | Ryan[Shield] | Train | Interceptor/Scarab | |
| 17728 | MenSol[Zero] | Move | (1148,2014),0,228,0 | 0 |
| 17730 | MenSol[Zero] | Move | (1145,2007),0,228,0 | 0 |
| 17732 | Ryan[Shield] | Move | (1409,1853),Hydralisk,228,0 | 3420 |
| 17734 | MenSol[Zero] | Attack Move | (1144,2002),0,228, | 0 |
| 17736 | Ryan[Shield] | Move | (1418,1872),Hydralisk,228,0 | 3410 |
| 17736 | MenSol[Zero] | Hotkey | Select,3 | |
| 17738 | MenSol[Zero] | Move | (1173,1950),0,228,0 | 0 |
| 17752 | MenSol[Zero] | Select | Hydralisk(x5) | 3452,3422,3393,5424,5499 |
| 17754 | Ryan[Shield] | Shift Select | 3475 | 3475 |
| 17756 | MenSol[Zero] | Move | (1008,1820),0,228,0 | 0 |
| 17758 | MenSol[Zero] | Move | (961,1861),0,228,0 | 0 |
| 17760 | MenSol[Zero] | Move | (1024,1897),0,228,0 | 0 |
| 17762 | Ryan[Shield] | Move | (1149,1891),Shuttle,228,0 | 3492 |

**Fig. 1.** Contents of a StarCraft Replay

While both actively recorded traces and those read from replays can share the same model, the content of such traces can be slightly different. The reason for this is that game replays are written by the game itself with direct access to game states while active recording is done through an interface on top of the game (Broodwar API, BWAPI) which abstracts from the underlying interactions of the player with the game.

This was one of the reasons why we chose not to work directly with primary traces in our application but with an already transformed version. Another reason is the way game replays are shown when they are examined with the means that the games provides. As their structure already suggests, replays are not simply played like a movie. Instead, the actions stored in them are executed within the game engine (much like in an actual game) and the results are shown. Therefore we can simply read replayed games through the same BWAPI interface that we use for active recording. Transforming a replay like this leads for instance to a “Move” command to a unit at one point in time being translated into a whole

number of attribute changes (x- and y-coordinates, velocity, angle etc.) for that unit over the next few game cycles.

The last and most important reason for performing preliminary trace transformations is the actual result we hope to achieve from using the information stored in these traces: We are aiming for a better performance of our ML agent trying to play parts of the game. While player interactions can certainly provide important information to this end, player actions are always only a reflection on what happens inside the game environment. By looking at those in-game states and omitting actions issued to units, the information on the human interaction with the game is translated into only in-game information. This is however the information that is most relevant for an AI agent learning how to play the game. Because of these considerations we chose to use in-game data. This in-game data is, according to the previous elaborations and also according to the theory behind TBR [3] a trace transformed by the game engine. However, unlike TBR theory, this transformation is not fully deterministic and also not reversible due to the complex nature of the computations inside the game engine. While traces can always be linked to their original replays, at the lowest level, i.e. the level at which we record actions from the game environment, there can be slight differences between transformed traces that were generated from the same replay making it non-deterministic.

A further decision we made, was to not record the entire game state at any one point in time, but only differences between the current and the previous state. The major reason for this choice was the far lower computational requirements while retaining the same amount of information. StarCraft, much like any RTS game, requires real-time actions. However, internally it runs through a set number of game cycles each second. At standard speed there are 24 cycles each second which would require huge amounts of information to be written to the database at any one cycle. If, on the other hand, only differences are recorded, these writing operations become a fraction of the previous amount. Currently for each unit 60 attributes are monitored. Each player can have up to 200 units. This would mean that 24K float values (192KB) per cycle or roughly 4.5 MB per second have to be written to the database. In an average game with a length of 20 minutes this can easily lead to 5GB of data. If we only record differences on the other hand, we can manage to store about 7.5h of gameplay in a 3GB database while retaining all information.

## 3 Trace Model

The model of the original primary traces gathered from user interactions is defined by the game itself which records all user interactions and saves them as replays of the game. Figure 1 shows an excerpt of a replay displayed in a third-party tool “BWChart” since replays are recorded in binary format.

We defined the trace model based on the BWAPI interface while focusing on “Units” as the important objects. This includes any changeable environment components, e.g. units and buildings. The actual observed elements (obsels)

[6] that make up the core of the trace are attribute changes in the previously mentioned Unit objects.

One example for an observed element is the change of the X value of the position of the monitored unit 33787 by -46 that can be seen in the first line of Figure 4.

Apart from the obsels, initial values of a game are also stored. These values such as map parameters and starting positions are only saved once since they are static, therefore no deltas for these values have to be recorded. These static values are omitted from the database model in Figure 2 for reasons of simplification.

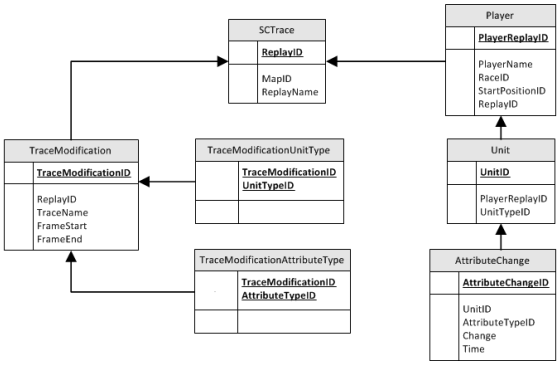


**Fig. 2.** Simplified Database Model for SC Traces

The model of the trace is similar to other trace models but more heavily focused on StarCraft specific data in specific and the problem domain in general. The structure is common to most RTS games and includes the previously mentioned obsels stored as "AttributeChange" that are linked to initial, unchanging, state of the environment in the "SCTrace" component. The "Trace Modification" part and its attachments represent the knowledge on trace transformations.

All of the data is stored in a relational database in order to facilitate efficient and easy access.

## 4 Trace Visualization and Transformation

Some features of StarTrace, like the possibility to display attribute changes over time in diagrams, were planned from the very start. Other features were added as demand arose from designing and implementing the agent that learns the game.

- Enable users to create new traces based on transformations applied to one or more existing traces.
- Provide access to all recorded information through visual means.

– Visualize the recorded traces in an easy to understand way that allows a top-down approach for recognizing patterns and episode signatures.
– Offer filtering opportunities to only display selected parts of one or several traces.

For now, the interface allows the expert to more easily identify salient features of traces, but the goal is ultimately to allow the agent to learn from its traces.This can either be done by improving the learning performance directly (finding and reusing episode signatures during the retrieval phase of cases) or by better understanding the intrinsic agent behavior by visualizing its performance over time.

### 4.1 Design and Implementation

StarTrace is a browser application based on PHP, HTML, CSS and AJAX. It uses the MySQL database which the traces are recorded to. Figure 3 shows the layout of the main window. StarTrace provides several filter options when searching through the primary traces. Primary traces in context of the tool are traces that have already been transformed from recorded actions to in-game attributes as described in Section 3. Once a primary trace has been selected, further transformations can be applied in order to filter through the trace. Users can select which types of obsels they want to look at by limiting both Unit objects that attributes are displayed for and attributes that are displayed.

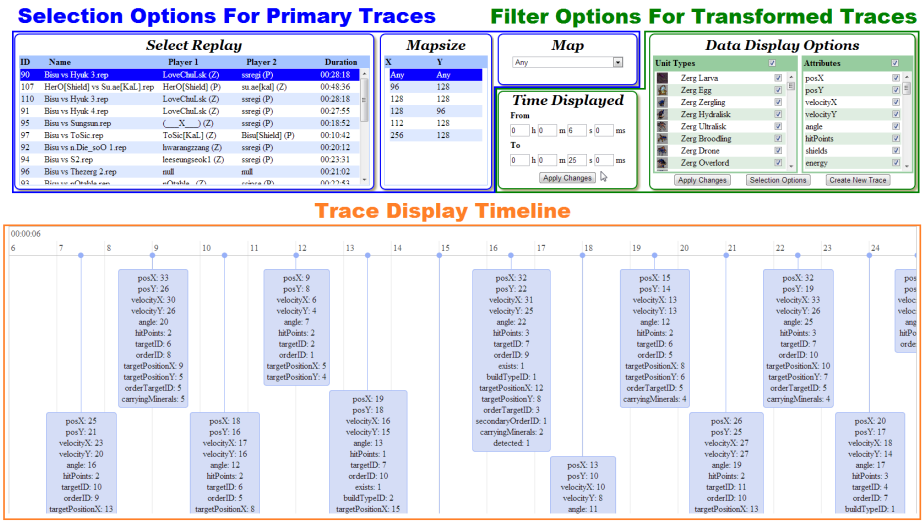


**Fig. 3.** Overview of the StarTrace Main GUI

The trace display window shows the result of the filter options selected: All changes of chosen attributes from the selected trace in the selected time frame

for chosen unit types. As the time frame can be freely modified between a few milliseconds and several minutes, the sheer amount of displayed information could easily have become confusing. Therefore, the standard view only gives a summary of the numbers and types of attributes that have changed in the selected time frame. Details on attribute differences can be seen when one of the summary elements is selected (Figure 4).

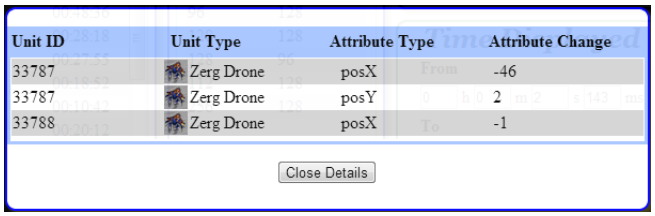


**Fig. 4.** Detailed View of Attribute Differences

A central feature of a TBS is the ability to define new traces based on existing ones. Currently, StarTrace allows the creation of new traces based on the filter criteria that are also available in the main view (Figure 5). Additionally we added a feature that is mostly based on the desire to enable the ML agent to learn from these traces: the ability to not use the differences of attribute values for each time step but to use aggregate values. In terms of unit locations this means for instance that the trace will not contain the change in position for a certain time step but the absolute position at each time step.

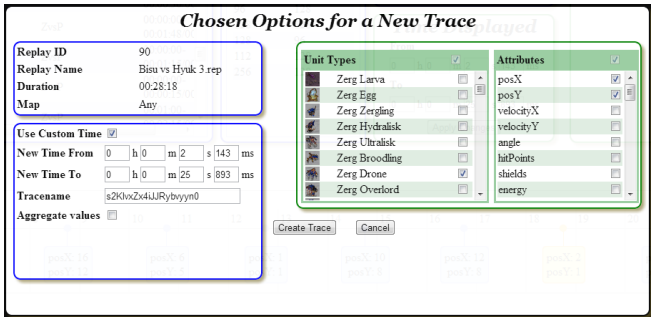


**Fig. 5.** GUI for Creating a New Trace

Section 4.2 showcases how an aggregate trace displayed in the tool can elaborate agent behavior inside the game and thus eventually enable improvements in the learning performance.

### 4.2 Using StarTrace to Understand Agent Behavior

Figure 6 shows an excerpt of a diagram display for an example transformed trace. The transformation resulted in an M-trace consisting of aggregate values instead of differences and only two of the 60 original attributes. Furthermore, the transformed trace only contains data for one specific type of unit.

For each individual unit, the development of the values for both attributes hitPoints (green) and groundWeaponCooldown (red) is shown over time.

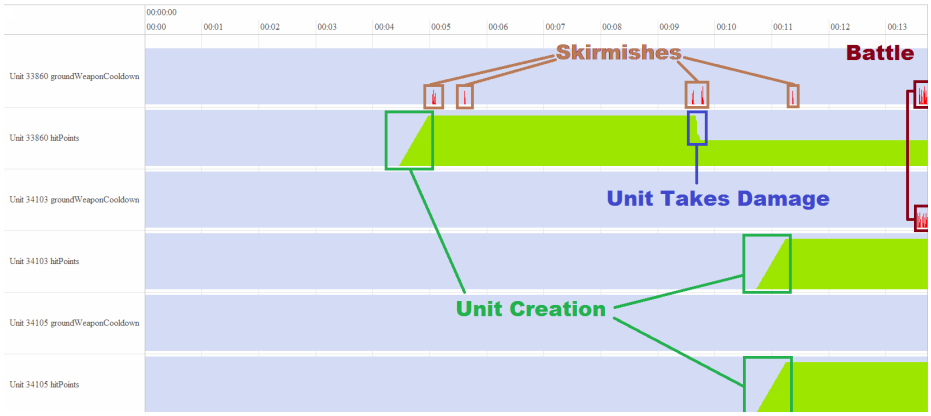


**Fig. 6.** Diagram of a Trace Transformed to Display Selected Aggregate Values (with Annotations)

The displayed trace can already show certain effects that are important to gameplay. On a strategic level, the rise of the hitPoint variable marks the creation of the respective units, i.e. the point in time when the player felt it necessary to build this type of unit. The short cycles where the groundWeaponCooldown variable first rises (after firing a weapon) and then decreases back to zero marks confrontations. As can be seen from the intensity, at first there is mostly short skirmishes. Towards the end of the displayed period there is a full scale battle in which two of the three displayed units are involved.

Distinguishing between skirmishes and full battles which both require different behavior would allow the AI agent to decide on different micromanagement behavior patterns. This makes sense as these two situations usually have different aims such as scouting/reconnaissance for skirmishes versus elimination of strategic groups of buildings or units for battles. However, elements like movement patterns and targeting strategies can not be learned from the values included in the currently displayed transformed trace. Therefore, after identifying these key events, a next step would now be to go back to the original M-Trace and create another transformation containing observed elements such as changes in x/y positions, velocities and target IDs.

## 5 Related Work

There are numerous approaches that try to use the intrinsic knowledge stored in StarCraft replays. [7] analyze a large corpus of StarCraft replays in the context of cognitive research. The authors try to find a correlation between actions that are observable in the replays and performing successfully in the games. Their results show that winning games is directly related to the number of actions that a player performs. In [8] the authors build a case base from automatically annotated StarCraft replays. They do this by defining a "Goals-to-win-StarCraft" ontology and automatically breaking up replays into cases by splitting them according to the actions happening.

[4] directly applies TBR to micromanagement in StarCraft in a way that is at least partially similar to what we are planning for our agent. The author creates an ML agent to micromanage combat unit based on CBR. Some traces of unit attributes are used in the case descriptions. However, the development process does not involve the analysis of game replays or a review of recorded agent behavior, attribute selection is only based on expert knowledge.

There are other systems to record, manage and transform traces that work similar to StarTrace. Georgon et al. [9] develop ABSTRACT, a tool to analyze users interactions with a complex technical device such as a car. ABSTRACT visualizes those traces and allows experts to define patterns in the user traces. The aim is to use the combination of expert knowledge and trace representation to define a cognitive model of the user that generates the trace. TStore [10] is a web-based TBMS which handles the storage, transformation, and reuse of modeled traces. Besides providing predefined options for transforming traces, it also offers the ability to define customized transformations based on Finite State Transducers. The authors test the trace recording performance of their environment by collecting user interaction traces from Wanaclip, a video clips composer.

## 6 Conclusion and Future Work

In this paper we presented StarTrace, a tool to visualize and transform traces of the RTS game StarCraft. The tool enables users to work with M-Traces generated from user interaction traces. Those user interaction traces are either user interaction traces originating from StarCraft itself or are player interactions recorded through an interface during active gameplay. StarTrace provides functionality to filter through and visualize StarCraft traces in a number of ways. It furthermore allows the user to create new traces by selecting filtering as well as transformation options for obsels directly in the GUI.

Our goal is to use StarTrace and traces in general to improve the performance of an ML agent that attempts to learn parts of the game. In an example application we showed how trace visualization and transformation provided through StarTrace can lead to better understanding of recorded performance in the game. Eventually we plan to re-integrate knowledge obtained through the tool directly

into the learning process of the agent.

There is a also number of other possible future improvements that have arisen from applying StarTrace to the transformed StarCraft traces. If unit attributes in a trace are not only selectable by unit type or attribute type but for each Unit object in a trace separately, that would give a maximum of control over the filtering options. Section 4.2 already showed how diagrams for several units and attributes from a trace can be displayed simultaneously. We plan on extending this to include the possibility to display several distinct traces at once, eventually with the option of merging values from multiple traces.

A crucial component of a TBS is the possibility to manually identify, store and elaborate on recognition patterns in traces. Among other things, externally appended annotations such as those in Figure 6 could then be added directly with the tool. This feature is the next major planned addition to the tool and will enable a whole new set of potential applications.

## References


1. Buro, M., Furtak, T.: Rts games and real-time ai research. In: Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS), Citeseer (2004) 63–70
2. Wender, S., Watson, I.: Applying reinforcement learning to small scale combat in the real-time strategy game starcraft:broodwar. In: Computational Intelligence and Games (CIG), 2012 IEEE Symposium on. (2012)
3. Mille, A.: From case-based reasoning to traces-based reasoning. Annual Reviews in Control **30**(2) (2006) 223–232
4. Szczepański, T.: Game ai: micromanagement in starcraft. Master's thesis, Norwegian University of Science and Technology (2010)
5. Georgeon, O., Henning, M.J., Bellet, T., Mille, A.: Creating cognitive models from activity analysis: A knowledge engineering approach to car driver modeling. In: International Conference on Cognitive Modeling. (2007) 43–48
6. Cordier, A., Mascret, B., Mille, A.: Dynamic case based reasoning for contextual reuse of experience. In: Provenance-Awareness in Case-Based Reasoning Workshop. ICCBR. (2010) 69–78
7. Lewis, J., Trinh, P., Kirsh, D.: A corpus analysis of strategy video game play in starcraft: Brood war. In: The Annual Meeting Of The Cognitive Science Society (COGSCI 2011). (2011)
8. Weber, B., Ontanón, S.: Using automated replay annotation for case-based planning in games. In: ICCBR Workshop on CBR for Computer Games (ICCBR-Games), Springer (2010)
9. Georgeon, O., Mille, A., Bellet, T.: Analyzing behaviorial data for refining cognitive models of operator. In: Database and Expert Systems Applications, 2006. DEXA'06. 17th International Workshop on, IEEE (2006) 588–592
10. Zarka, R., Champin, P.A., Cordier, A., Egyed-Zsigmond, E., Lamontagne, L., Mille, A.: Tstore: A web-based system for managing, transforming and reusing traces. In Luc Lamontagne, J.A.R.G., ed.: ICCBR 2012 TRUE and Story Cases Workshop. (September 2012) 173–182