An Improved Dataset and Extraction Process for Starcraft AI

Glen Robertson and Ian Watson
Department of Computer Science
University of Auckland
Auckland, New Zealand, 1010
{glen, ian}@cs.auckland.ac.nz

Abstract

In order to experiment with machine learning and data mining techniques in the domain of Real-Time Strategy games such as StarCraft, a dataset is required that captures the complex detail of the interactions taking place between the players and the game. This paper describes a new extraction process by which game data is extracted both directly from game log (replay) files, and indirectly through simulating the replays within the StarCraft game engine. Data is then stored in a compact, hierarchical, and easily accessible format. This process is applied to a collection of expert replays, creating a new standardised dataset. The data recorded is enough for almost the complete game state to be reconstructed, from either player's viewpoint, at any point in time (to the nearest second). This process has revealed issues in some of the source replay files, as well as discrepancies in prior datasets. Where practical, these errors have been removed in order to produce a higher-quality reusable dataset.

1 Introduction

Games are an ideal domain for exploring the capabilities of Artificial Intelligence (AI) within a constrained environment and a fixed set of rules, where problem-solving techniques can be developed and evaluated before being applied to more complex real-world problems (Schaeffer 2001). Ideally, increasingly realistic games will also lead to more human-like AI being developed (Laird and van Lent 2001). Board game AI has historically received a lot of academic and public attention, but over the past decade there has been increasing interest in research based on video game AI.

Real-Time Strategy (RTS) is a genre of video games in which players indirectly control many units in a simplified military simulation, which usually includes gathering resources, building infrastructure and armies, and managing units in battle. RTS games present some of the toughest challenges for AI agents, making it a difficult area for developing competent AI (Buro and Furtak 2004). It is a particularly attractive area for AI research because of how human players can quickly become adept at dealing with the complexity of the game, with experienced humans outplaying even the best academic agents (Buro and Churchill 2012).

RTS games have huge state spaces and delayed rewards, so heuristic-based search techniques, which have proven effective in a range of board games (Schaeffer 2001), have difficulty with anything but the most restricted subproblems of RTS AI. Many researchers in the field have sought to deal with this challenge by examining the actions taken by human players, using techniques based around keyhole plan recognition (Dereszynski et al. 2011; Hsieh and Sun 2008; Synnaeve and Bessière 2011) or learning from demonstration (Ontañón et al. 2008; Palma et al. 2011; Weber, Mateas, and Jhala 2012). StarCraft[1] is a very popular RTS game which has recently been increasingly used as a platform for AI research. Due to the popularity of StarCraft, there are many expert players available to provide knowledge and examples of play, producing plentiful information for researchers.

Most RTS games can save a game log (replay) file when a match ends, and expert players often upload their replays to websites for others to watch. In StarCraft, a replay file records the starting conditions and player actions in a match, allowing the entire match to be played back as a deterministic simulation within the game engine. This makes for very compact replay files, but means that game state information is not directly available. In order to apply machine learning or data mining to StarCraft data, researchers usually need to run a simulation and extract the relevant information. This creates a time-consuming hurdle for each new researcher, therefore a comprehensive and accessible dataset, suitable for a wide range of applications, is needed.

This paper starts by outlining the existing work related to extracting and using data from StarCraft replay files, demonstrating the need for a better extraction method and dataset than is currently available. Next it gives the main goals for producing the dataset, followed by the design of the extraction process and data recording used to meet those goals. This is followed by a detailed description of the dataset and what is recorded. An evaluation of the resulting dataset is carried out, comparing it to the previous best data available, leading to a conclusion on whether the dataset meets the specified goals and is an improvement on prior work. Finally, areas of future work and improvements are identified.

[1] Blizzard Entertainment: StarCraft: `blizzard.com/games/sc/`

2 Related Work

A number of papers have focused on extracting information from StarCraft replay files, even in the relatively short time since interest began to grow in using StarCraft as a research platform. Before then, the RTS games used for research purposes, such as Wargus[2] and ORTS[3], lacked the expert player base and wide availability of replays to make the approach worthwhile. Information is usually extracted for analysis, such as determining common strategies, and for creating or evaluating computer-controlled players (bots). In many cases, machine learning algorithms are applied to predict a player's strategic choices given the (often incomplete) information known at an earlier point in time. When applied to a bot, this approach can be used to predict opponent actions and to select actions for the bot itself.

To the authors' knowledge, the first published work focusing on data extraction from player replays in StarCraft was Hsieh and Sun (2008). They used an existing tool to convert the player actions and their timings – stored in replay files found on a popular StarCraft site – into readable textual log files. Because the replay file does not store game states, basic state information was inferred based on the construction actions taken by the players. A case base and state lattice were created for each of the game's three "races", allowing the prediction of strategies and analysis of the popularity and effectiveness of build orders (the orders in which buildings are constructed in a game).

Weber and Mateas (2009) followed a similar route, downloading a set of over 5400 replay files from popular StarCraft sites, and using an existing tool to extract player actions into textual log files. However, in this case each resultant log was labeled with a strategy based on expert-defined rules for the build order. This labeled data was used to train classifiers to predict the labeled strategy with missing or noisy information, as well as to train regression algorithms to predict the timing of certain actions.

Later, a similar process was undertaken in Churchill and Buro (2011), Dereszynski et al. (2011), and Hostetler et al. (2012). Again, each went through the process of collecting replay files from websites, however, this time the Brood War Application Programming Interface (BWAPI) was used to connect to StarCraft while playing back the replays as a simulation, allowing for much more complete state information to be extracted. However, each still focused on strategic-level build order information, recording numbers of units and buildings in existence every 21 or 30 seconds. In Churchill and Buro (2011) the information was used for comparison with their own build order planner, while in Dereszynski et al. (2011) and Hostetler et al. (2012) it was used to train models for strategy analysis and prediction. Wender, Cordier, and Watson (2013) also extracted replay files through BWAPI, however this time focusing on low-level unit control (micromanagement) and visualisation and transformation of replay data.

The most similar work to our work is Synnaeve and Bessiere (2012), as it focused on producing a reusable dataset and extraction process, as well as carrying out extraction, analysis and machine learning processes like the other work outlined here. They collected over 8000 replays from popular StarCraft sites, and filtered out many problematic files to result in a set of 7649 replays. The replay files were simulated within StarCraft and information was recorded to three separate text files per replay. Although this work was a useful contribution to the field, some issues remain. Firstly, it is tuned to high-level (strategic) information, so it records only the position attributes of units from over a hundred possible attributes, and stores this only every hundred game frames – approximately every four seconds – providing insufficiently fine-grained data for examining mid-level (tactical) or low-level (micromanagement) activities. Secondly, due to a limitation of BWAPI, it cannot record the actual actions taken by players, but instead must watch for changes in in-game unit orders and try to filter out changes which were not the result of player actions, resulting in discrepancies between the true actions and those seen in output. Thirdly, the output format of three text files, two of which store multiple different types of data in different sections, makes parsing and using the data an arduous process, particularly if searching the data for particular pieces of information.

Recently, Cho, Kim, and Cho (2013) again followed a very similar process of replay downloading and extraction as in Weber and Mateas (2009), but this time used BWAPI to additionally extract the unit visibility events. This provided enough information to determine which opponent units and buildings each player knew about throughout the game, taking into account the fact that the game limits player visibility to an area surrounding their own units. Strategy and victory prediction was then carried out both with and without the limited information.

Extracting information from the immense quantities of expert knowledge encoded in the form of StarCraft replay files is clearly an area of high interest within the field of RTS game AI, yet, until recently, each researcher was forced to reinvent the wheel with a new extractor in order to glean the data they require from the encoded replay files. Synnaeve and Bessiere (2012) sought to move the field away from this repetition and unnecessary work, but the dataset is not flexible or fine-grained enough to be able to be used for machine learning at all of the different levels of granularity seen within StarCraft. This work seeks to address these issues.

3 Goals and Approach

In order to create an improved standard StarCraft dataset which builds on Synnaeve and Bessiere (2012) and yet is appropriate for the full range of research in StarCraft AI, four major goals were identified: completeness and accuracy of the information stored, and accessibility and extensibility of the dataset and extraction process itself.

For the information to be complete and accurate, the extractor will need to capture as much useful data about the game state as possible, from a wide range of replays, to provide a much more complete rendering of the available information than other datasets. With this level of detail, the user of the dataset should be able to reconstruct the complete

[2]Wargus: `wargus.sourceforge.net`

[3]Open RTS: `skatgame.net/mburo/orts`

game state at any point in the game, from either player's viewpoint. The dataset should become usage agnostic, instead of being aimed at just high- or low-level play, as the fine-grained detail can be used, abstracted, or ignored as required.

Over 7500 professional-level matches were analysed, using the same set of replay files used in Synnaeve and Bessiere (2012) for consistency and comparability. Player actions in the matches were recorded by directly parsing replay files, allowing the true player actions to be extracted, including unit groupings used. This approach simplifies the action extraction (ignoring the complexity in the external code used to parse replay files) and makes it simple to identify observers (non-player participants) in a match early in the extraction process, because they have few actions. In a separate process, game states throughout the matches were recorded by simulating the matches within StarCraft and reading the state using BWAPI (figure 1). All unit attributes are recorded, making this a complete representation of the state.

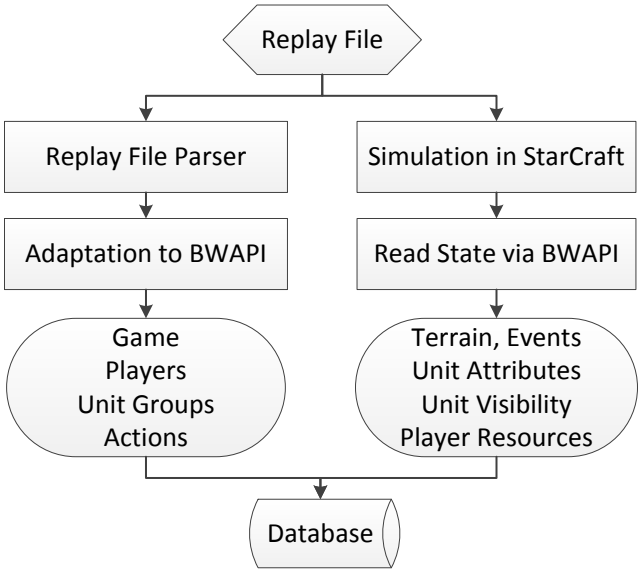

Figure 1: Overview of the extraction process.

To be accessible and extensible, the dataset must obviously be far easier to read than the StarCraft replay files, and ideally should be easier to read than the text format used in prior work. It should enable quick access to information about states without requiring scanning of an entire match's information, so that a user can efficiently find states of interest. It should also be able to be altered or updated easily, and the extraction process re-run relatively quickly, so that a user can modify the extraction process and update the result instead of waiting for a (lengthy) full extraction run. Finally, the output should be as compact as possible so that the extracted data from many replays may be stored and examined or downloaded by new users.

A database-centred design was chosen to allow for structured data to be stored and accessed quickly with a well-known query language. The hierarchical and referential data inherent in RTS games – for example, each unit must belong to a player – can be effectively represented using tables with foreign keys. Databases also provide powerful indexing capability for fast lookup of information even in large datasets, so that game state information about a particular subset of features at a particular point in time can be retrieved easily and efficiently. To reduce the recording size, only changes in game state are recorded. Additionally, it is possible to skip frames in order to trade off accuracy for accessibility (in file size). Appropriate indices allow the most recent value of an attribute to be retrieved efficiently even when the actual time it changed is unknown, and they also facilitate updating of entries, so the extraction process can be re-run quickly. Using the indices and relational information, the extractor can check for unwanted entries and remove them during the extraction process. If the process is to be altered to store more data, it is simple to add additional rows, columns, or tables as desired.

4 Extraction Method

The data stored represents interactions over time between players and the game, recording static player and terrain information, as well as dynamic player actions, resources, events, unit attributes and visibility in a database. A careful method was devised to process the replays consistently and without introducing errors.

First, the replay name and duration (in game frames), along with the names, actions, and in-game "races" of the players are parsed from the replay file. Before the information is stored in the database, the actions are processed as follows.

1. Control groups – used by players to store and retrieve a selection of units using number keys – are replaced with regular unit selection actions. A limitation here is that dead units cannot be filtered out of the unit groups at this point, as unit status is not stored in the replay, so some actions will be incorrectly recorded as if issued to groups in which some or all units are dead (not possible in the game).
2. Consecutive unit selection actions are removed except for the final selection, since unit selection actions in StarCraft have no effect on the state except when followed by a non-selection action.
3. Players with the fewest actions are removed until only two remain, as matches often have additional players who are actually observing the match, but have to join as participants due to a limitation in StarCraft. An additional check is made to ensure none of the excluded players performed many actions compared to the included players.
4. A winner is determined if the recording shows one player leaving the game before the other (not always).

At this point, the information can be stored in the database. Selection actions are used only to identify the units that were selected and the groups in which they were selected, so that the non-selection actions performed with these unit groups may be stored.

For the remaining information, the replay is loaded in StarCraft and accessed through BWAPI. First, static map information is recorded, including the name and number of player starting positions, as well as buildability, walkability, ground height, and region identifier of each map tile. This static information could equivalently be read from the replay file, but is more easily accessible through BWAPI. In order to ease spatial reasoning, instead of simply storing a list of narrow openings between two map areas (choke points), base locations and start locations, a walking distance measure to the nearest choke point, base location, and start location is stored with each map tile.

Next, dynamic game information is recorded as the match is simulated. By default, changes are recorded every in-game second (24 frames) to limit the amount of space required while still providing four times the resolution of prior work – enough to capture in full detail everything except precise micromanagement reactions. If changes are recorded every frame approximately eight times more space is required – this tradeoff is discussed further in the next two sections. The extractor records changes to all unit attributes accessible through BWAPI, changes in unit visibility from each player's perspective, and changes in resources and supply (population limit) held by each player, enabling a complete view of the game state to be reconstructed from either player's perspective for any given second in the game. Additional information is recorded for convenience, as it is mostly derivable from the change information stored above. This includes in-game events such as units being created and destroyed, or changing type (redundant), players leaving, and nuclear launches being detected (non-redundant). It also includes a set of aggregate region values stored for each player, summing the value of ground units, air units, buildings, and resources of which they are aware, for themselves and the enemy, in that region.

Notably, the unit visibility information recorded is vital to reconstructing a game state as a player would see it in-game, as a player's vision of the map is limited to areas near their own units. Prior work has almost always ignored the visibility of units, as it cannot be extracted from the replay files directly, making it impossible to tell which unit movements (or other attribute changes) each player is aware of. Ignoring visibility limitations makes strategy prediction challenges vastly easier, as most of the hidden information in the game derives from units and buildings which are hidden from a player. Only Cho, Kim, and Cho (2013) and Hostetler et al. (2012) address this issue, as they were specifically examining strategy inference with limited information. Synnaeve and Bessiere (2012) records the first time a unit or building is seen, but doesn't record subsequent changes in visibility.

5 Adaptive Granularity

A challenge when recording information in a game as complex as StarCraft is the tradeoff between information granularity and storage space. Storing all of the game state information every frame – even just the changes – is costly in terms of space, yet fine-grained information can be important to playing the game. This is particularly true in the realm of micromanagement, in which professional players quickly and carefully control individual or small groups of units to maximise their effectiveness, usually in combat. In order to better handle this potential use of the dataset, experimentation was carried out to evaluate two potential new ways to adapt the granularity, which we refer to as attack-based adaptation and action-based adaptation.

Attack-based adaptation builds on the basic fixed interval recording, by recording the game state at fixed intervals but reducing the intervals during attacks. It uses the same base frame-rate as the default recording method, but records four times more frequently during combat (as determined by any unit attacking or being attacked). This rate was chosen because it equates to a very high rate of 240 effective actions per minute, similar to that of the fastest players in the world, and therefore should capture all of the detail seen in player behavior. A potential drawback of this method is that it cannot distinguish between attacks which require fast player control, such as a large battle, from those that do not, such as a turret automatically firing at nearby enemies. Likewise, it cannot detect other non-attack situations in which fast control is needed.

With action-based adaptation, frame recording happens each time a player makes an action instead of being time-based. This means that fewer frames per second are recorded when players don't need to make many decisions, such as at the start of the game, while more frames per second are recorded when players are rapidly controlling many units and buildings, such as during the intense later stages of the game. Another benefit of this approach is that it stores the exact state the game was in when a player made an action, which could help to detect reactions to changes in state. However, occasionally – particularly early in a match, when few actions are being made – it could actually hinder detection of changes because non-action states are not recorded. This drawback could potentially be mitigated by requiring a minimum recording frame rate in situations where few actions are made.

6 Evaluation

In addition to the expected advantages of greatly increased information accuracy and faster querying, the described method of extracting and storing replay data yields some unexpected findings when compared with prior methods. Firstly, it is possible to identify corrupted replays which occur due to a replay being recorded in an older version of StarCraft. In these replays, the rules of the game have changed between recording and playback, causing the simulation to increasingly deviate from the correct state. By comparing the units in the replay file with those seen in the game, 3751 of the 7660 replays were identified as containing invalid units, although 668 of those replays had fewer than 1% invalid units. All replays with more than 1% invalid units were removed from the final dataset.

Comparing the player actions recorded directly from replay files to those recorded in-game in previous work, the higher fidelity of the new recording method becomes evident. By referring to the actual unit groupings used by the player, far fewer orders are recorded, despite the orders showing greater detail and better representing actual player

actions. Certain player actions that don't correspond to unit orders, such as setting an exit point for a factory, are now recorded. Additionally, unit order changes that don't correspond to player actions, such as automatically attacking a nearby enemy, are no longer recorded as if they were player actions. This comparison has also helped to identify likely errors in the previous action recording, as certain actions appear to be repeated multiple times in the recording.

The extraction method described in this paper and the adaptive granularity alternatives were evaluated on a test dataset consisting of the games in which both players chose the "Protoss" race – one of the six possible race matchups. The unit attribute changes form the vast majority of the data, averaging 96% of the total size of the test dataset, so it is worthwhile to examine these attribute changes further. Looking at the proportion of attribute changes per unit type (figure 2), we see that originally, 61% of attribute change records are related to "probe" worker units, which is by far the highest proportion of any unit. Worker units move around automatically and are fairly numerous, so their attribute changes take up a substantial amount of space, yet they are rarely involved in combat or other micromanagement. Therefore, action-based adaptation was applied to individual workers, recording their attribute changes less frequently unless they had recently been given an action. This change reduced them to to 30% of attribute changes, and reduced the overall dataset size by a similar proportion.

Looking at attribute changes per attribute (figure 3), we see that 15% of attribute changes record an order timer, and a further 22% (total) record angle and velocity information. Based on domain knowledge, these attributes are unlikely to be important for most analysis and probably could be filtered out completely, while the position attributes are much more likely to be important. However, in the interests of keeping the dataset as complete as possible, these attributes have remained in the dataset.

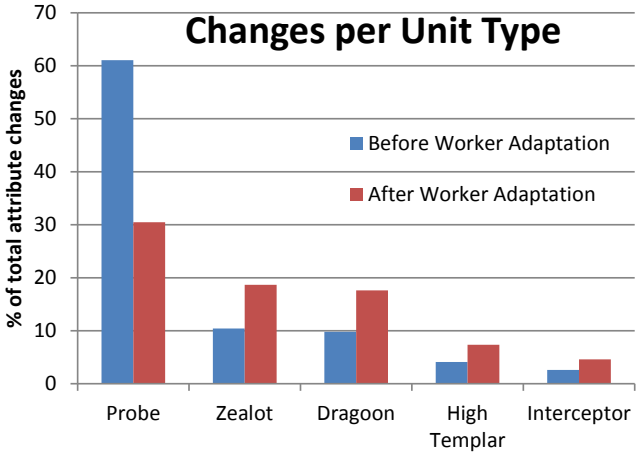

Figure 2: Frequency of attribute changes grouped by unit type, showing top 5. Using data from the test dataset recorded at fixed intervals of 24 frames.

Finally, we may compare the effects of the adaptive granularity methods (figure 4). There is clearly a tradeoff between

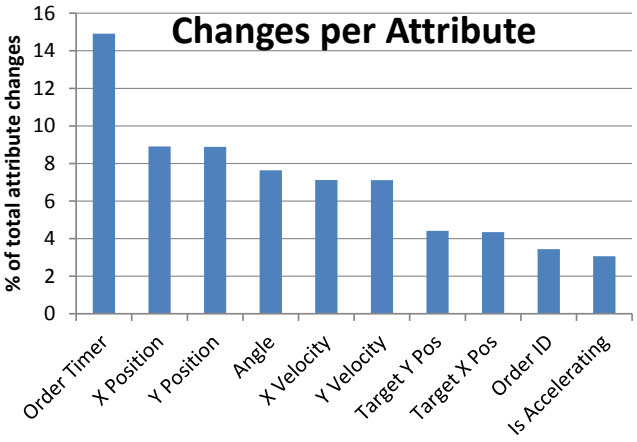

Figure 3: Frequency of attribute changes grouped by attribute, showing top 10. Using data from the test dataset recorded at fixed intervals of 24 frames.

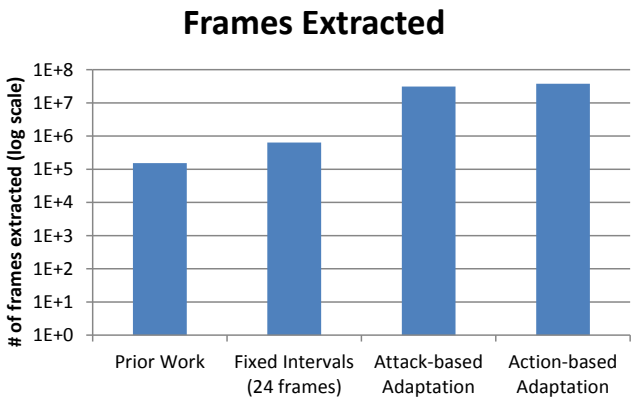

Figure 4: Number of frames recorded by each extraction method. Prior work refers to Synnaeve and Bessiere (2012). Using data from the test dataset.

accuracy and size, but it is difficult to determine whether this tradeoff is worthwhile for a general case. The fixed interval extraction is able to capture sufficient information to understand all but the most fast-paced decisions, and the adaptive granularity methods should cover even those situations. However, the number of frames extracted increases by over an order of magnitude when using either of the adaptive granularity methods, and the storage space required approximately doubles. Given the already large size of the dataset – multiple gigabytes for just the fixed interval extraction of the test dataset – the adaptive granularity methods will not be used for the final dataset. However, because the dataset can be relatively quickly modified by re-running the extractor, it can still easily be customised to particular needs.

7 Conclusions and Future Work

This paper has presented a new method for extracting StarCraft replay data for machine learning and data mining. The method combines the strengths of two different information

sources: direct parsing of replay file data and simulation of replay data within the StarCraft game engine. By directly parsing replay files, we are able to accurately record the actual actions the players made, instead of watching for the actions' effects, and we can much more easily identify corrupted replay files. By simulating the replays in the game, we can record the complete set of unit attributes, including visibility information, so that the game state at any point can be reconstituted. This produces complete and accurate data, especially compared with prior work, which recorded at most one quarter of the frame rate and just a few of the approximately one hundred unit attributes.

In addition, the paper describes an effective structure for storing the data such that it is easily accessible and extensible. The source code for the extractor is available[4] so that further extensions and modifications can be made.

Three methods were tested which varied the choice of frames to extract: extracting frames at fixed intervals, extracting at fixed intervals but with a higher rate during attacks, and extracting frames whenever players made actions. For the full extraction process of a standardised dataset[5], the simplest, fixed interval extraction method was used, because it provides a comprehensive recording, which should be sufficient for anything except precise micromanagement analysis. If more fine-grained analysis is required, the standard dataset is easily modified by reducing the interval or using an adaptive granularity method.

Although not used in the full extraction process, the adaptive granularity extraction methods showed promise for data of widely varying levels of abstraction, and may prove useful in other fields. They could be better optimised by restricting the fine-grained information recording spatially and contextually, instead of just temporally. For example, when using attack-based adaptation, the extra information could be recorded only for units nearby to those involved in the attack, and when using action-based adaption, the extra information could be recorded just for units that were included in the action. However, these sorts of optimisations require more domain knowledge to implement well, and are thus difficult to generalise.

Acknowledgements

Special thanks to Stefan Wender for the original database design built upon in this work.

References

Buro, M., and Churchill, D. 2012. Real-time strategy game competitions. *AI Magazine* 33(3):106–108.

Buro, M., and Furtak, T. M. 2004. RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference*, 63–70. Citeseer.

Cho, H.-C.; Kim, K.-J.; and Cho, S.-B. 2013. Replay-based strategy prediction and build order adaptation for StarCraft AI bots. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 329–335.

Churchill, D., and Buro, M. 2011. Build order optimization in StarCraft. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment (AIIDE) Conference*, 14–19.

Dereszynski, E.; Hostetler, J.; Fern, A.; Dietterich, T.; Hoang, T.; and Udarbe, M. 2011. Learning probabilistic behavior models in real-time strategy games. In *Proceedings of the AIIDE Conference*, 20–25. AAAI Press.

Hostetler, J.; Dereszynski, E.; Dietterich, T.; and Fern, A. 2012. Inferring strategies from limited reconnaissance in real-time strategy games. In *Proceedings of the Annual Conference on Uncertainty in Artificial Intelligence*, 367–376.

Hsieh, J., and Sun, C. 2008. Building a player strategy model by analyzing replays of real-time strategy games. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, 3106–3111. Hong Kong, China: IEEE.

Laird, J., and van Lent, M. 2001. Human-level AI's killer application: Interactive computer games. *AI Magazine* 22(2):15–26.

Ontañón, S.; Mishra, K.; Sugandh, N.; and Ram, A. 2008. Learning from demonstration and case-based planning for real-time strategy games. In Prasad, B., ed., *Soft Computing Applications in Industry*, volume 226. Springer Berlin / Heidelberg. 293–310.

Palma, R.; Sánchez-Ruiz, A.; Gómez-Martín, M.; Gómez-Martín, P.; and González-Calero, P. 2011. Combining expert knowledge and learning from demonstration in real-time strategy games. In Ram, A., and Wiratunga, N., eds., *Case-Based Reasoning Research and Development*, volume 6880 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 181–195.

Schaeffer, J. 2001. A gamut of games. *AI Magazine* 22(3):29–46.

Synnaeve, G., and Bessière, P. 2011. A bayesian model for plan recognition in RTS games applied to StarCraft. In *Proceedings of the AIIDE Conference*, 79–84. AAAI Press.

Synnaeve, G., and Bessiere, P. 2012. A dataset for StarCraft AI and an example of armies clustering. In *Proceedings of the AIIDE Workshop on AI in Adversarial Real-Time Games*.

Weber, B., and Mateas, M. 2009. A data mining approach to strategy prediction. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 140–147. IEEE.

Weber, B.; Mateas, M.; and Jhala, A. 2012. Learning from demonstration for goal-driven autonomy. In *Proceedings of the AAAI Conference on AI*, 1176–1182.

Wender, S.; Cordier, A.; and Watson, I. 2013. Building a trace-based system for real-time strategy game traces. In *Proceedings of the International Conference on Case-Based Reasoning (ICCBR) Workshop on Experience Reuse: Provenance, Process-Orientation and Traces*.

[4]Data extractor code available at: `github.com/phoglenix/ScExtractor`

[5]Dataset available at: `www.cs.auckland.ac.nz/research/gameai/projects.php`