StarCraft Strategy Extraction

IRTM Practical Assignment

Dennis Soemers
Student ID: I6037052

Course: Information Retrieval & Text Mining
Maastricht University
Department of Knowledge Engineering
Faculty of Humanities and Sciences

June 3, 2015

1 Introduction

StarCraft: Brood War is a complex, real-time strategy game that is a popular testbed for real-time game AI research (Ontañón et al., 2013). Every year, the game is used in multiple different competitions, such as the AIIDE StarCraft AI Competition, where AI programmers can submit their intelligent game-playing agents to participate. The majority of the participants in these competitions have little strategic diversity, and strategies are generally hardcoded. In this context, a strategy can be seen as a high-level plan to execute throughout the entire game, and it does not concern low-level, short-term decision making. Because hardcoding such strategies is a time-consuming process, most agents only use one or a small number of strategies. For this reason, there is also very little research in recognizing the strategy that the opponent executes during a game, and adapting the played strategy to counter the opponents'.

This report describes a first attempt at automatically extracting strategies for the game of *StarCraft: Brood War* from strategy pages on a domain-specific wiki. If strategies can automatically be extracted and serialized to, for instance, XML files, an AI programmer only needs to implement a parser to read such a strategy and can relatively easily enable his agent to play any number of different strategies.

The remainder of this report is structured as follows. Section 2 describes previous, related work. In Section 3, the corpus that has been used is described. Section 4 explains how the data has been pre-processed. A description of the main strategy extraction process is provided in Section 5. Section 6 explains how a user can navigate through the files output by the strategy extraction process, and how the results can be visualized to gain more insight. Finally, Section 7 concludes the report and provides ideas for future work.

2 Related Work

To the best of the author's knowledge, there is only one publication that describes work where text was automatically processed with the aim of improving the performance of a game-playing agent (Branavan, Silver, & Barzilay, 2011). In this work, features extracted from text are combined with features directly taken from the game state in a multi-layer neural network, and the parameters of the network are learned via Monte-Carlo simulations. The domain in which this technique was used is the turn-based game *Civilization II*.

Because this technique learns from the text in an online manner, and requires the ability to simulate (larger parts of) games, it is not easily applicable in the game of *StarCraft*. This technique also does not explicitly learn full strategies in a way that they can be explained to the user. This means that the user can not verify that the obtained results are correct or fix small mistakes, and therefore requires the results to be completely accurate for them to be useful. Due to these properties of the technique, it has not been employed for this assignment, and the choice was made to use more conventional information retrieval and

text mining approaches.

3 Corpus

The corpus consists of 181 strategy pages that have been taken from the page http://wiki.teamliquid.net/starcraft. This website is a popular and well-respected site in the community of *StarCraft* players. The corpus contains a small number of duplicate documents, but this is no problem because they are (mostly) processed separately anyway. The documents contain the full HTML code of the original pages, meaning that the HTML tags can be used in addition to the text during the strategy extraction process. All of the pages are relatively small; the entire process described in the remainder of the report can be run on the full corpus in a few seconds on a standard PC. None of the data has been manually annotated, and, to the best of the author's knowledge, there is no alternative data available for the domain of *StarCraft* that is already annotated. See Figure 1 for an example of the wiki pages included in the corpus.

Figure 1: Example of a wiki page

In the process described in the following sections of the report, it is assumed that every document in the corpus describes a single strategy. Furthermore, it is assumed that a strategy can be characterized by (1) one or more *Build Orders*, and (potentially empty) lists of (2) other strategies that are *hard* or *soft counters* to the described strategy, (3) other strategies to which the described strategy is a *hard* or *soft counter*, and (4) *maps* on which the described strategy is *strong* or *weak*. The meaning of these game-specific terms is explained in Section 5. In truth, some of the documents in the corpus do not describe a strategy, but these will simply be ignored when the expected information is not found in them. Documents that describe strategies also often contain more information than only the elements listed above, such as a description of the history of a strategy (when and by whom it was popularized). Because the

goal of the project is serialize strategies to an easily parsable format such as XML, there needs to be a restriction on the content that is extracted, and the elements described above have been chosen as content that will be extracted and serialized.

4 Pre-processing

The first step in pre-processing a document is to collect those HTML elements in the document that can potentially be relevant for the strategy extraction process, and discarding everything else. All elements that do not have one of the following HTML tag names are immediately discarded; *h1, h2, h3, h4, h5, h6, ul, ol, p.* In this way, all headers, lists and paragraphs (of text) are kept, and other elements (such as scripts) are discarded. Of the remaining elements, those that have one of the following values for their *id* attribute are also discarded; *catlinks, footer, column-one, toc.* These elements correspond to areas of the website that are not relevant for the strategy page (for instance, *column-one* corresponds to the column visible on the left-hand side in Figure 1), or to areas with redundant information (for instance, *toc* corresponds to the table of contents, which essentially duplicates information that can already be found by looking at all the headers on a page). In this step, the text "[edit]" is also removed from any element where it is found at the end of the element's text (this text corresponds to the buttons that can be used to edit the wiki page).

In the second pre-processing step, the different sizes of different header elements are exploited to create a tree-based representation of the document. The relevant elements (which, by now, can only be headers, lists or paragraphs) are stored in a tree, where header elements correspond to internal nodes and list and paragraph elements correspond to leaf nodes. Smaller headers are always placed below larger headers in the tree, and elements that appear closer to the top of the webpage will always be placed to the left in the tree of siblings that appear closer to the bottom of the webpage. Figure 2 depicts an example of a part of a document and the corresponding tree-based representation. This structure can later be used to, for instance, conveniently find all the headers that describe what a certain list or paragraph of text is about. For example, in the document shown in Figure 2, the list starting with "3 Hatch Muta" is a list of strategies that are soft counters to the "2 Rax FE (vs. Zerg)" strategy. So, the headers "2 Rax FE (vs. Zerg)", "Countered By", and "Soft Counters" (the ancestor nodes) are relevant for this list, but the "Build Order" header (not an ancestor node) is not relevant for this list.

5 Strategy Extraction

5.1 Leaf Node Classification

The first step in the strategy extraction process is to classify the content of all the leaf nodes of the tree built as described in the previous section. Every

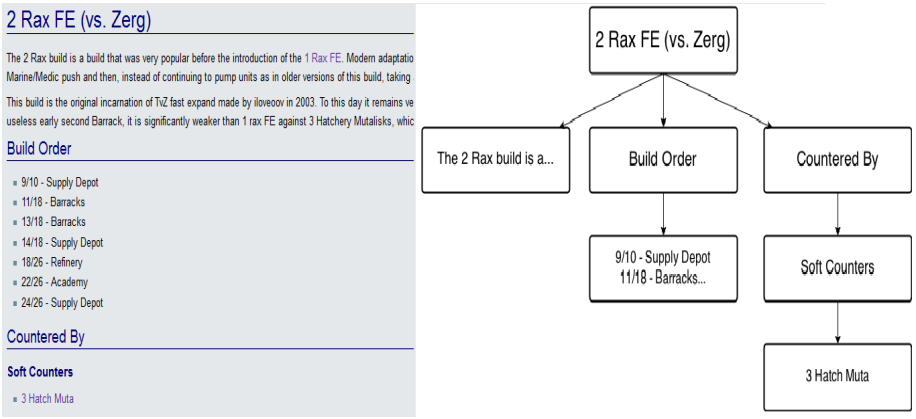

Figure 2: Example of part of a document and the corresponding tree-based representation

leaf node can be classified as containing one of the types of content that we are interested in finding as described in Section 3 (*Build Order, Countered By Soft, Countered By Hard, Counter To Soft, Counter To Hard, Strong Maps, Weak Maps*), or *Unknown* if is not recognized as any of the other content types. Internal nodes, which are nodes that correspond to headers, are not classified, but the text of these headers is used in the classification process of leaf nodes.

Due to the lack of annotated data, this classification is performed using simple, handcrafted rules. Given a leaf node l, it is classified as follows. Let $H(l)$ be the set of all the headers contained in nodes that are ancestors of l. $H(l)$ is then searched for a number of specific terms, and if certain combinations of terms are found in $H(l)$, l is classified accordingly. See Table 1 for a complete overview of all the terms that are searched for and the corresponding classifications. The table also shortly describes what every classification means in terms of the game of *StarCraft*.

The *Build Order* classification can only be used for list elements, whereas the other classifications can also be applied to text paragraph elements. All of the involved searches for substrings are performed with the text of headers converted to contain only lowercase letters. The headers are searched for terms in the order that they appear in the table. So, if $H(l)$ contains the terms "map", "strong" and "weak" as substrings, l will be classified as describing strong maps. In practice, no cases have been found where this order matters, but not all documents have been manually checked. Some cases have been manually found where lists were incorrectly classified as *Build Orders* that were actually just lists providing background information. Any list or paragraph of text that cannot be classified according to the rules given above, is simply classified as "Unknown" and not used in the remainder of the process (though a potential use of these elements is suggested in Section 7). Once the content of

Table 1: Header terms and corresponding leaf classification

Set of Terms	**Leaf Classification**	**Description**
{map, strong}	Strong Maps	Maps on which this is a strong strategy
{map, weak}	Weak Maps	Maps on which this is a weak strategy
{countered, by, hard}	Countered By (Hard)	Strategies that are hard counters to this strategy
{countered, by}	Countered By (Soft)	Strategies that are soft counters to this strategy
{counter, to, hard}	Counter To (Hard)	Strategies to which this strategy is a hard counter
{counter, to}	Counter To (Soft)	Strategies to which this strategy is a soft counter
{build}	Build Order	Build Order (list of instructions to execute during gameplay)

every leaf node has been classified, it is processed in a different way depending on the classification.

5.2 Build Order Extraction

Build Orders are the most important parts of the documents that we are interested in extracting. Build Orders can be viewed as lists of instructions that should be executed during gameplay, and these instructions essentially define a strategy (whereas the other content types only add information, such as explaining in which situations a strategy can be strong or weak). Build Orders are typically formatted on the wiki as a list, where every list element is a single instruction. A single instruction generally has one or more preconditions (game-state conditions that, when satisfied, signal that the instruction should be executed) and one or more actions to take. Generally, these actions are to construct a building or train a new unit of a certain type, but in rare cases it can also be a different instruction (for instance, telling the player to send a unit out to scout the map).

See Figure 3 for an example of a (very well-structured) Build Order from one of the documents. In this example, every instruction is formatted consistently as "⟨Current Supply⟩/⟨Max Supply⟩ - ⟨Building Type to construct⟩". The "⟨Current Supply⟩/⟨Max Supply⟩" part can (loosely) be interpreted as specifying a point in time during a game, and in fact only the first of the two numbers

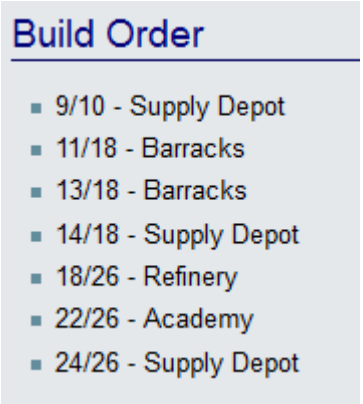

Figure 3: Example of a Build Order

is important. There are also, sometimes, preconditions in different formats, but they can generally be easily extracted with handcrafted rules.

Any part of a single instruction's text that is not recognized as one of the known patterns for preconditions, is assumed to describe an action to take. The Lemmatizer, Tokenizer and POS Tagger of the Stanford CoreNLP toolkit (Manning et al., 2014) with their default settings are used to pre-process this text by splitting it up in tokens, lemmatizing the tokens, and assigning POS tags to the tokens.

Every verb that is found in this text is lemmatized, and then compared to a manually constructed mapping from verb tokens to known game actions, and if such a verb exists as a key in the mapping, the corresponding action is registered as being an action that should be executed according to the strategy. Currently, the mapping only contains the verb "scout", which is interpreted as describing that the player should send a unit out to scout the map. It is important to use the POS Tagger to be able to only detect "scout" as an action when it has been tagged as being a verb, because the same word can also be used to refer to the unit type named "Scout" when it is not a verb.

From the remainder of an instruction's text, all words and biwords are looked up in a handcrafted mapping of known building and unit type names. When type names are found, it is assumed that the instruction is to build one instance of the type. Most build orders (see, for instance, Figure 3) only mention the type name and do not explicitly say that a unit or building of that type should be constructed, so in most cases there is no need to perform a more complex analysis to find out what should be done with the recognized type. In rare cases, an instruction explicitly mentions to stop the production of a type, or to build a specific quantity (greater than one) of a type. These cases are currently not handled correctly. The reason for using biwords in addition to single tokens is that many unit and building type names consist of two tokens, but there

are none that have more than two tokens. The manually created mapping also contains commonly used abbreviations, in addition to the formal names of types (for instance, "goon" will be correctly recognized as "Dragoon").

5.3 Counter Relation Extraction

Another important part of the strategy extraction process is to extract "counter" relations between strategies. The leaf node classification described above can, in most cases, identify lists that describe other strategies that are soft or hard counters to the strategy described in the document, and lists to which the described strategy is a soft or a hard counter.

Sometimes, a page describes other strategies that have such a counter relation in natural language. These cases are difficult to deal with because they require a deeper understanding of the semantics of the text, and are currently not handled yet. In other cases, some related strategies are explicitly mentioned by name. Isolating these strategy names from the surrounding text is essentially the problem of Named Entity Recognition (NER).

Naturally, the default Stanford NER module (Finkel, Grenager, & Manning, 2005) has not been trained to handle this highly domain-specific entity type. The module also has an option to be trained on new data, and this has been attempted with a small amount of manually annotated data (1 document for training and 1 for testing), but the performance was very disappointing. This approach may be succesful with more time spent on manually annotating a much larger dataset.

Some examples of strategy names are; *1 Rax FE, 2 Gateway, 2 Port Wraith, 14 CC, Double Expand, Proxy 5 Rax.* There is not a consistent pattern or structure that is followed by all strategy names, so it is difficult to extract them all correctly with a rule-based system. The only obvious pattern is that strategy names are consistently written on the wiki using only numeric tokens and tokens starting with a capital letter. For this reason, the current implementation considers every sequence of consective tokens that are either numeric or start with a capital as a potential strategy name. Such potential token sequences are then compared to lists of known *StarCraft* game terms (such as unit or building names), and if they are not found in those lists they are accepted as a strategy name. This works fairly well because the system only attempts to extract strategy names from parts that have already been classified as describing counter relations, and these parts are often lists where every list element is just a single strategy name, but it is the least robust part of the system with the most room for improvement.

5.4 Map Extraction

The previous subsection described the problem of finding strategy names in parts of a document that have already been classified as listing certain types of counter relations. As described earlier, some parts of each document can also be classified as listing maps on which a strategy is strong or weak, and in these

parts of the document we want to search for maps in a very similar way to the problem of extracting strategy names described above.

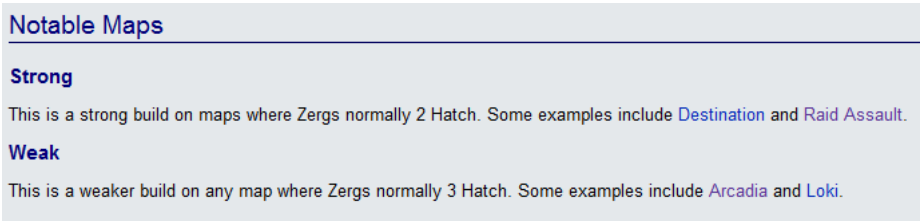

Figure 4: Part of a document that describes strong and weak maps

Figure 4 depicts two (very short) paragraps that have been classified as describing strong and weak maps. For both of these paragraphs, the first sentence provides a general description of a map in natural language. As described above for the case of descriptions of strategies in natural language, these cases are not dealt with yet because they require a more complex analysis of the text.

However, the second sentence of each paragraph in the example show that maps are sometimes mentioned by name. Extracting these could again be viewed as a Named Entity Recognition problem, as was also the case for strategy names. The important difference is that an exhaustive list of map names is available for the game, whereas this is not the case for strategy names. Therefore, map names can easily be extracted from parts of the document that have been classified as containing information on maps by taking all sequences of up to 5 tokens (because the longest map names have 5 tokens), and testing whether these sequences of tokens occur in a list of known map names.

6 Navigation & Visualization

The goal of most Information Retrieval applications is to extract information and structure, summarize or visualize the information so that it can be used by a human end user. The main goal of the application described in this report is a bit different, in that it is intended to write the extracted information to XML so that it can directly be used by other software (a game-playing agent). For this purpose, an application has been developed that can read pages from the wiki that have been saved to .txt or .html files and output corresponding .xml files. The source code and the Maven *pom.xml* file, as well as the entire corpus and the .xml files corresponding to the entire corpus, have been included as an attachment with the report.

However, because the application is known to not process every input completely as intended and makes mistakes, it is useful for an AI programmer to be able to compare the results to the original wiki page and manually check for errors. Even though this means that the output of the application cannot be used directly and does not completely eliminate the large amount of manual

work involved in creating new strategies for a game-playing agent to play, it still reduces the amount of manual work by a large portion. Processing the documents and manually checking for and correcting errors takes a lot less time than manually implementing entire strategies. For this purpose, the developed application contains a simple GUI that, when processing a single document (as opposed to a larger corpus), provides a side-by-side view with the extracted results on the left-hand side and the text of the original wiki page on the right-hand side. This GUI is depicted in Figure 5. Ideally, it would also be possible to edit the output and correct mistakes in this GUI, but this has not yet been implemented.

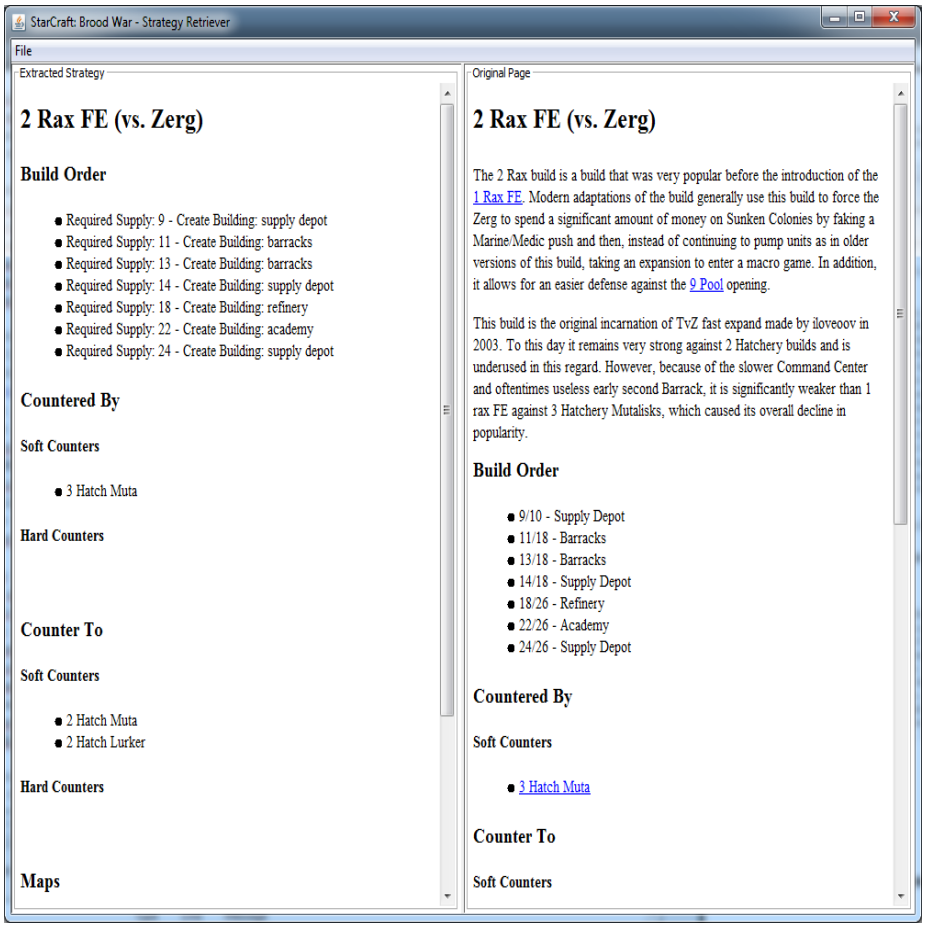

Figure 5: The application's GUI with an example document

Additionally, the counter relations that are extracted when processing a full corpus can also be used to construct a directed graph, where every vertex

represents a single strategy, and an edge from v to u denotes that v is a counter to u. Such a graph can then be visualized and provide more insight into the different strategies. A graph has been created in this way using the entire corpus, and visualized using a variety of layouts using GEPHI. Figure 6 (a) depicts the resulting graph using the Radial Axis Layout, with nodes grouped by In Degree and ordered by Out Degree. This layout immediately reveals two interesting sets of strategies, namely those with an in degree of 0 (no known counters) and those with an out degree of 0 (not known to be a counter to anything). Using the Force Atlas layout combined with Noverlap on the same graph produces the result visualized in Figure 6 (b). This figures reveals some clusters of strategies, where the wiki pages of strategies in the same cluster mention each other as counters, and documents corresponding to strategies in different clusters have no information on each other. In both figures, the color of every node indicates the race in the game for which the strategy was written (green for *Protoss*, blue for *Terran*, pink for *Zerg*). The .gexf file used to draw the graphs in GEPHI is also included with the report.

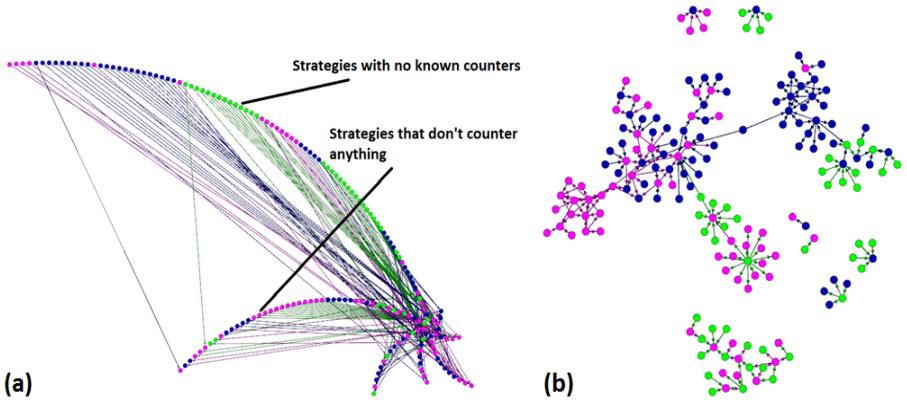

Figure 6: (a) Radial Axis Layout. Nodes grouped by: In Degree. Nodes ordered by: Out Degree. (b) Force Atlas Layout, followed up with Noverlap layout

7 Conclusions & Future Work

The report described an initial attempt at automatically extracting strategies for the game of *StarCraft: Brood War* from wiki pages and writing the relevant information in a highly structured format so that it can easily be parsed and used by an intelligent game-playing agent. All of the strategy extraction steps described in Section 5 make use of handcrafted rules. This is currently difficult to avoid due to the lack of annotated data and the large amount of time required for manually constructing annotated data. In most cases, these handcrafted solutions provide reasonable results.

It is difficult to provide exact metrics or a more formal analysis of the quality of the results, again due to the lack of a test set containing the "correct" results. Manual inspection of the results has indicated that most documents contain a few small errors that could be corrected relatively easily by hand, with some documents containing no errors and very few documents having completely wrong results. A tool with a GUI has been developed that allows for easy inspection of the results. It has also been shown that Gephi can be used to visualize the countering relations between strategies and provide insight in these relations. The current graphs should, however, be interpreted very carefully, since they highly depend on the process of extracting strategy names, which has been identified as producing the least satisfying results of all the different steps.

The most obvious idea for future work is to replace the handcrafted rules in the various steps of the strategy extraction process with more robust methods, such as machine learning algorithms. These will first require the construction of a manually annotated dataset. Another idea for future research is to perform sentiment analysis on some of the larger paragraphs of surrounding text that are present in many documents, but are currently ignored. This could provide additional insight into the strengths and weaknesses of strategies.

References

Branavan, S. R. K., Silver, D., & Barzilay, R. (2011). Learning to Win by Reading Manuals in a Monte-Carlo Framework. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1* (pp. 268–277).

Finkel, J. R., Grenager, T., & Manning, C. (2005). Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics* (pp. 363–370).

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. (2014, June). The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of the 52nd annual meeting of the association for computational linguistics: System demonstrations* (pp. 55–60). Baltimore, Maryland: Association for Computational Linguistics.

Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., & Preuss, M. (2013). A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *Computation Intelligence and AI in Games, IEEE Transactions on*, *5*(4), 293–311.