Combining Strategic Learning and Tactical Search in Real-Time Strategy Games

Nicolas A. Barriga and **Marius Stanescu** and **Michael Buro**
Department of Computing Science
University of Alberta, Canada
{barriga|astanesc|mburo}@ualberta.ca

Abstract

A commonly used technique for managing AI complexity in real-time strategy (RTS) games is to use action and/or state abstractions. High-level abstractions can often lead to good strategic decision making, but tactical decision quality may suffer due to lost details. A competing method is to sample the search space which often leads to good tactical performance in simple scenarios, but poor high-level planning.

We propose to use a deep convolutional neural network (CNN) to select among a limited set of abstract action choices, and to utilize the remaining computation time for game tree search to improve low level tactics. The CNN is trained by supervised learning on game states labelled by *Puppet Search*, a strategic search algorithm that uses action abstractions. The network is then used to select a script — an abstract action — to produce low level actions for all units. Subsequently, the game tree search algorithm improves the tactical actions of a subset of units using a limited view of the game state only considering units close to opponent units.

Experiments in the μRTS game show that the combined algorithm results in higher win-rates than either of its two independent components and other state-of-the-art μRTS agents.

To the best of our knowledge, this is the first successful application of a convolutional network to play a full RTS game on standard game maps, as previous work has focused on sub-problems, such as combat, or on very small maps.

1 Introduction

In recent years, numerous challenging research problems have attracted AI researchers to using real-time strategy (RTS) games as test-bed in several areas, such as case-based reasoning and planning (Ontañón et al. 2007), evolutionary computation (Barriga, Stanescu, and Buro 2014), machine learning (Synnaeve and Bessière 2011), deep learning (Usunier et al. 2017; Foerster et al. 2017; Peng et al. 2017) and heuristic and adversarial search (Churchill and Buro 2011; Barriga, Stanescu, and Buro 2015). Functioning AI solutions to most RTS sub-problems exist, but combining those doesn't come close to human level performance[1].

To cope with large state spaces and branching factors in RTS games, recent work focuses on smart sampling of the search space (Churchill and Buro 2013; Ontañón 2017; 2016; Ontañón and Buro 2015) and state and action abstractions (Uriarte and Ontañón 2014; Stanescu, Barriga, and Buro 2014; Barriga, Stanescu, and Buro 2017b). The first approach produces strong agents for small scenarios. The latter techniques work well on larger problems because of their ability to make good strategic choices. However, they have limited tactical ability, due to their necessarily coarse-grained abstractions. One compromise would be to allocate computational time for search-based approaches to improve the tactical decisions, but this allocation would come at the expense of allocating less time to strategic choices.

We propose to train a deep convolutional neural network (CNN) to predict the output of *Puppet Search*, thus leaving most of the time free for use by a tactical search algorithm. *Puppet Search* is a strategic search algorithm that uses action abstractions and has shown good results, particularly in large scenarios. We will base our network on previous work on CNNs for state evaluation (Stanescu et al. 2016), reformulating the earlier approach to handle larger maps.

This paper's contributions are a network architecture capable of scaling to larger map sizes than previous approaches, a policy network for selecting high-level actions, and a method of combining the policy network with a tactical search algorithm that surpasses the performance of both individually.

The remainder of this paper is organized as follows: Section 2 discussed previous related work, Section 3 describes our proposed approach and Section 4 provides experimental results. We then conclude and outline future work.

2 Related Work

Ever since the revolutionary results in the *ImageNet* competition (Krizhevsky, Sutskever, and Hinton 2012), CNNs have been applied successfully in a wide range of domains. Their ability to learn hierarchical structures of spatially invariant local features make them ideal in settings that can be represented spatially. These include uni-dimensional streams in natural language processing (Collobert and Weston 2008), two-dimensional board games (Silver et al. 2016), or three-dimensional video analysis (Ji et al. 2013).

These diverse successes have inspired the application of CNNs to games. They have achieved human-level perfor-

[1]`http://www.cs.mun.ca/~dchurchill/starcraftaicomp/report2015.shtml#mvm`

Table 1: Input feature planes for Neural Network. 25 planes for the evaluation network and 26 for the policy network.

Feature	# of planes	Description
Unit type	6	Base, Barracks, worker, light, ranged, heavy
Unit health points	5	$1, 2, 3, 4$, or ≥ 5
Unit owner	2	Masks to indicate all units belonging to one player
Frames to completion	5	$0-25$, $26-50$, $51-80$, $81-120$, or ≥ 121
Resources	7	1, 2, 3, 4, 5, $6-9$, or ≥ 10
Player	1	Player for which to select strategy

mance in *Atari* games, by using Q-learning, a well known reinforcement learning (RL) algorithm (Mnih et al. 2015). But the most remarkable accomplishment may be AlphaGo (Silver et al. 2016), a *Go* playing program that last year defeated Lee Sedol, one of the top human professionals, a feat that was thought to be at least a decade away. As much an engineering as a scientific accomplishment, it was achieved using a combination of tree search and a series of neural networks trained on millions of human games and self-play, running on thousands of CPUs and hundreds of GPUs.

These results have sparked interest in applying deep learning to games with larger state and action spaces. Some limited success has been found in micromanagement tasks for RTS games (Usunier et al. 2017), where a deep network managed to slightly outperform a set of baseline heuristics. Additional encouraging results were achieved for the task of evaluating RTS game states (Stanescu et al. 2016). The network significantly outperforms other state-of-the-art approaches at predicting game outcomes. When it is used in adversarial search algorithms, they perform significantly better than using simpler evaluation functions that are three to four orders of magnitude faster.

Most of the described research on deep learning in multi-agent domains assumes full visibility of the environment and lacks communication between agents. Recent work addresses this problem by learning communication between agents alongside their policy (Sukhbaatar, Szlam, and Fergus 2016). In their model, each agent is controlled by a deep network which has access to a communication channel through which they receive the summed transmissions of other agents. The resulting model outperforms models without communication, fully-connected models, and models using discrete communication on simple imperfect information combat tasks. However, symmetric communication prevents handling heterogeneous agent types, limitation later removed by (Peng et al. 2017) which use a dedicated bi-direction communication channel and recurrent neural networks. This would be an alternative to the search algorithm we use for the tactical module on section 4.3, in cases where there is no forward model of the game, or there is imperfect information.

A new search algorithm that has shown good results particularly in large RTS scenarios, is *Puppet Search* (Barriga, Stanescu, and Buro 2015; 2017a; 2017b). It is an action abstraction mechanism that uses fast scripts with a few carefully selected choice points. These choice points are then exposed to an adversarial look-ahead procedure, such as

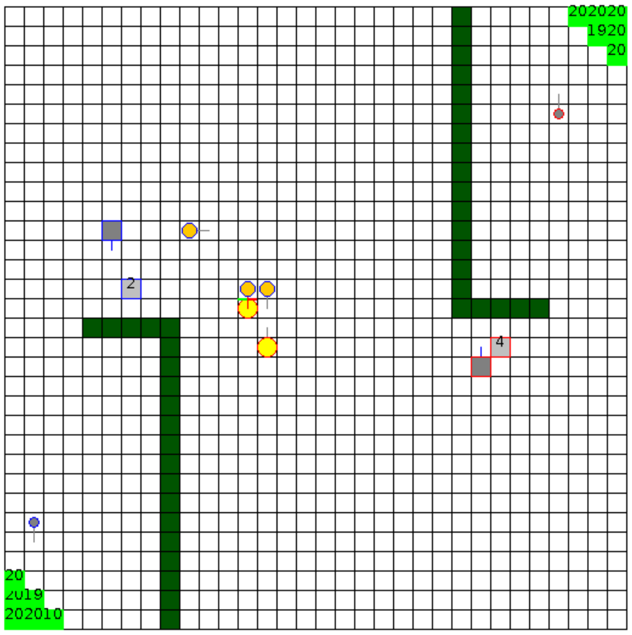

Figure 1: μRTS screenshot from a match between scripted LightRush and HeavyRush agents. Light green squares are resources, dark green are walls, dark grey are barracks and light grey the bases. Numbers indicate resources. Grey circles are worker units, small yellow circles are light combat units and big yellow ones are heavy combat units. Blue lines show production, red lines an attack and grey lines moving direction. Units are outlined blue (player 1) and red (player 2). μRTS can be found at `https://github.com/santiontanon/microrts`.

Alpha-Beta or Monte Carlo Tree Search (MCTS). The algorithm then uses a forward model of the game to examine the outcome of different choice combinations and decide on the best course of action. Using a restricted set of high-level actions results in low branching factor, enabling deep look-ahead and favouring strong strategic decisions. Its main weakness is its rigid scripted tactical micromanagement, which led to modest results on small sized scenarios where good micromanagement is key to victory.

3 Algorithm Details

We build on previous work on RTS game state evaluation (Stanescu et al. 2016) applied to μRTS (see figure 1).

Table 2: Neural Network Architecture

Evaluation Network	Policy Network
Input 128x128, 25 planes	Input 128x128, 26 planes
2x2 conv. 32 filters, pad 1, stride 1, LReLU Dropout 0.2	
3x3 conv. 32 filters, pad 0, stride 2, LReLU Dropout 0.2	
2x2 conv. 48 filters, pad 1, stride 1, LReLU Dropout 0.2	
3x3 conv. 48 filters, pad 0, stride 2, LReLU Dropout 0.2	
2x2 conv. 64 filters, pad 1, stride 1, LReLU Dropout 0.2	
3x3 conv. 64 filters, pad 0, stride 2, LReLU Dropout 0.2	
1x1 conv. 64 filters, pad 0, stride 1, LReLU	
1x1 conv. 2 filters pad 0, stride 1, LReLU	1x1 conv. 4 filters pad 0, stride 1, LReLU
Global averaging over 16x16 planes	
2-way softmax	4-way softmax

This study presented a neural network architecture and experiments comparing it to simpler but faster evaluation functions. The CNN-based evaluation showed a higher accuracy at evaluating game states. In addition, when used by state-of-the-art search algorithms, they perform significantly better than the faster evaluations. Table 1 lists the input features their network uses.

The network itself is composed of two convolutional layers followed by two fully connected layers. It performed very well on 8×8 maps. However, as the map size increases, so does the number of weights on the fully connected layers, which eventually dominates the weight set. To tackle this problem, we designed a fully convolutional network (FCN) which only consists of intermediate convolutional layers (Springenberg et al. 2014) and has the advantage of being an architecture that can fit a wide range of board sizes.

Table 2 shows the architectures of the evaluation network and the policy network we use, which only differ in the first and last layers. The first layer of the policy network has an extra plane which indicates which player's policy it is computing. The last layer of the evaluation network has two outputs, indicating if the state is a player 1 or player 2 win, while the policy network has four outputs, each corresponding to one of four possible actions. The global averaging used after the convolutional layers does not use any extra weights, compared to a fully connected layer. The benefit is that the number of network parameters does not grow when the map size is increased. This allows for a network to be quickly pre-trained on smaller maps, and then fine-tuned on the larger target map.

Puppet Search requires a forward model to examine the outcome of different actions and then choose the best one. Most RTS games do not have a dedicated forward model or simulator other than the game itself. This is usually too slow to be used in a search algorithm, or even unavailable due to technical constraints such as closed source code or being tied to the graphics engine. Using a policy network for script selection during game play allows us to bypass the need for a forward model of the game. Granted, the forward model is still required during the supervised training phase, but execution speed is less of an issue in this case, because training is performed offline. Training the network via reinforcement learning would remove this constraint completely.

Finally, with the policy network running significantly faster (3ms versus a time budget of 100ms per frame for search-based agents) than *Puppet Search* we can use the unused time to refine tactics. While the scripts used by *Puppet Search* and the policy network represent different strategic choices, they all share very similar tactical behaviour. Their tactical ability is weak in comparison to state-of-the-art search-based bots, as previous results (Barriga, Stanescu, and Buro 2017b) suggest.

For this reason, the proposed algorithm combines an FCN for strategic decisions and an adversarial search algorithm for tactics. The strategic component handles macro-management: unit production, workers, and sending combat units towards the opponent. The tactical component handles micro-management during combat.

The complete procedure is described by Algorithm 1. It first builds a limited view of the game state, which only includes units that are close to opposing units (line 2). If this limited state is empty, all available computation time is assigned to the strategic algorithm, otherwise, both algorithms receive a fraction of the total time available. This fraction is decided empirically for each particular algorithm combination. Then, in line 9 the strategic algorithm is used to compute actions for all units in the state, followed by the tactical algorithm that computes actions for units in the limited state. Finally, the actions are merged (line 11) by replacing the strategic action in case both algorithms produced actions for a particular unit.

4 Experiments and Results

All experiments were performed in machines running Fedora 25, with an Intel Core i7-7700K CPU, with 32GB of

Algorithm 1 Combined Strategy and Tactics

```
 1: procedure GETCOMBINEDACTION($state, stratAI,$
                                $tactAI,$
                                $stratTime,$
                                $tactTime$)
 2:     $limState \leftarrow$ EXTRACTCOMBAT($state$)
 3:     if ISEMPTY($limState$) then
 4:         SETTIME($stratAI, stratTime + tactTime$)
 5:     else
 6:         SETTIME($stratAI, stratTime$)
 7:         SETTIME($tactAI, tactTime$)
 8:     end if
 9:     $stratActions \leftarrow$ GETACTION($stratAI, state$)
10:     $tactActions \leftarrow$ GETACTION($tactAI, limState$)
11:     return MERGE($stratActions, tactActions$)
12: end procedure
```

RAM and an NVIDIA GeForce GTX 1070 with 8GB of RAM. The Java version used for μRTS was OpenJDK 1.8.0, Caffe git commit 365ac88 was compiled with g++ 5.3.0, and pycaffe was run using python 2.7.13.

The *Puppet Search* version we used for all the following experiments utilizes alpha-beta search over a single choice point with four options. The four options are *WorkerRush*, *LightRush*, *RangedRush* and *HeavyRush*, and were also used as baselines in the following experiments.

Two other recent algorithms were also used as benchmarks, Na¨veMCTS (Ontañón 2013) and Adversarial Hierarchical Task Networks (AHTNs) (Ontañón and Buro 2015). Na¨veMCTS is an MCTS variant with a sampling strategy that exploits the tree structure of Combinatorial Multi-Armed Bandits — bandit problems with multiple variables. Applied to RTS games, each variable represents a unit, and the legal actions for each of those units are the values that each variable can take. NaïveMCTS outperforms other game tree search algorithms on small scenarios. AHTNs are an alternative approach, similar to *Puppet Search*, that instead of sampling from the full action space, uses scripted actions to reduce the search space. It combines minimax tree search with HTN planning.

All experiments were performed on 128x128 maps ported from the *StarCraft: Brood War* maps used for the AIIDE competition. These maps, as well as implementations of *Puppet Search*, the four scripts, AHTN and NaïveMCTS are readily available in the μRTS repository.

4.1 State Evaluation Network

The data for training the evaluation network was generated by running games between a set of bots using 5 different maps, each with 12 different starting positions. Ties were discarded, and the remaining games were split into 2190 training games, and 262 test games. 12 game states were randomly sampled from each game, for a total of 26,280 training samples and 3,144 test samples. Data is labelled by a Boolean value indicating whether the first player won. All evaluation functions were trained on the same dataset.

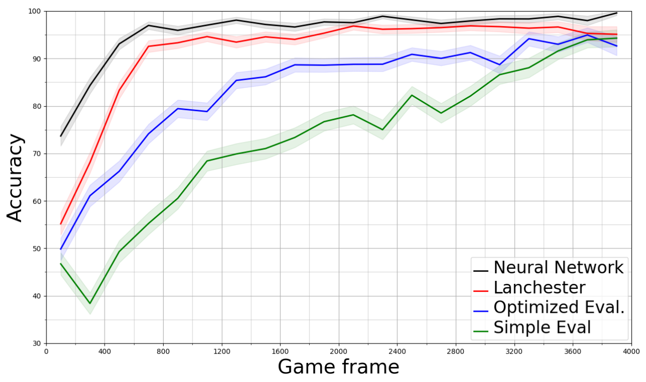

Figure 2: Comparison of evaluation accuracy between the neural network and the built-in evaluation function in μRTS. The accuracy of predicting the game winner is plotted against game time. Results are aggregated in 200-frame buckets. Shaded areas represent one standard error.

The evaluation network reaches 95% accuracy in classifying samples as wins or losses. Figure 2 shows the accuracy of different evaluation functions as game time progresses. The functions compared are the evaluation network, Lanchester (Stanescu, Barriga, and Buro 2015), the simple linear evaluation with hard-coded weights that comes with μRTS, and a version of the simple evaluation with weights optimized using logistic regression. The network's accuracy is even higher than previous results in 8x8 maps (Stanescu et al. 2016). The accuracy drop of the simple evaluation in the early game happens because it does not take into account units currently being built. If a player invests resources in units or buildings that take a long time to complete, its score lowers, despite the stronger resulting position after their completion. The other functions learn appropriate weights to mitigate this issue.

Table 3 shows the performance of *PuppetSearch* when using the Lanchester evaluation function and the neural network. The performance of the network is significantly better (P-value = 0.0011) than Lanchester's, even though it is three orders of magnitude slower. Evaluating a game state using Lanchester takes an average of 2.7μs, while the evaluation network uses 2,574μs.

Table 4 shows the same comparison, but with *Puppet Search* searching to a fixed depth of 4, rather than having 100ms per frame. The advantage of the neural network is much more clear, as execution speed does not matter in this case. (P-value = 0.0044)

4.2 Policy Network

We used the same procedure as in the previous subsection, but now we labelled the samples with the outcome of a 10 second *Puppet Search* using the evaluation network. The resulting policy network has an accuracy for predicting the correct puppet move of 73%, and a 95% accuracy for pre-

Table 3: Evaluation network versus Lanchester: round-robin tournament using 60 different starting positions per match-up and **100ms of computation time.**

	PS CNN	PS Lanc.	Light Rush	Heavy Rush	Avg.
PS CNN	-	59.2	89.2	72.5	73.6
PS Lanc.	40.8	-	64.2	67.5	57.5
LightRush	10.8	35.8	-	71.7	39.4
HeavyRush	27.5	32.5	28.3	-	29.4

Table 4: Evaluation network versus Lanchester: round-robin tournament on 20 different starting positions per match-up, searching to **depth 4.**

	PS CNN	PS Lanc.	Light Rush	Heavy Rush	Avg.
PS CNN	-	80	95	82.5	85.8
PS Lanc.	20	-	82.5	90	64.2
LightRush	5	17.5	-	70	30.8
HeavyRush	17.5	10	30	-	19.2

Table 5: Policy network versus Puppet Search: round-robin tournament using 60 different starting positions per match-up.

	Policy Net.	PS	Light Rush	Heavy Rush	Ranged Rush	Worker Rush	Avg.
Policy net.	-	44.2	94.2	71.7	100	61.7	61.9
PS	55.8	-	87.5	66.67	91.7	93.3	65.8
LightRush	5.8	12.5	-	71.7	100	100	48.3
HeavyRush	28.3	33.3	28.3	-	100	100	48.3
RangedRush	0	8.3	0	0	-	100	18.1
WorkerRush	38.3	6.7	0	0	0	-	7.5

Table 6: Mixed Strategy/Tactics agents: round-robin tournament using 60 different starting positions per match-up.

	Policy Naïve	PS Naïve	PS	Policy Network	Light Rush	Heavy Rush	Ranged Rush	AHTN P	Worker Rush	Naïve MCTS	Avg.
Policy net.-Naïve	0.0	56.7	97.5	100.0	100.0	95.8	100.0	72.5	74.2	98.3	88.3
PS-Naïve	43.3	0.0	81.7	79.2	90.0	94.2	93.3	90.0	90.8	93.3	84.0
PS	2.5	18.3	0.0	63.3	86.7	69.2	92.5	96.7	95.0	93.3	68.6
Policy net.	0.0	20.8	36.7	0.0	94.2	71.7	100.0	57.5	61.7	97.5	60.0
LightRush	0.0	10.0	13.3	5.8	0.0	71.7	100.0	100.0	100.0	96.7	55.3
HeavyRush	4.2	5.8	30.8	28.3	28.3	0.0	100.0	100.0	100.0	74.2	52.4
RangedRush	0.0	6.7	7.5	0.0	0.0	0.0	0.0	100.0	100.0	86.7	33.4
AHTN-P	27.5	10.0	3.3	42.5	0.0	0.0	0.0	0.0	64.2	68.3	24.0
WorkerRush	25.8	9.2	5.0	38.3	0.0	0.0	0.0	35.8	0.0	71.7	20.6
NaïveMCTS	1.7	6.7	6.7	2.5	3.3	25.8	13.3	31.7	28.3	0.0	13.3

dicting any of the top 2 moves.

Table 5 shows the policy network coming close to *Puppet Search* and defeating all the scripts.

4.3 Strategy and Tactics

Finally, we compare the performance of the policy network and *Puppet Search* as the strategic part of a combined strategic/tactical agent. We will do so by assigning a fraction of the allotted time to the strategic algorithm and the remainder to the tactical algorithm, which will be NaïveMCTS in our experiments. We expect the policy network to perform better in this scenario, as it runs significantly faster than *Puppet Search* while maintaining similar action performance.

The best time split between strategic and tactical algorithm was determined experimentally to be 20% for *Puppet Search* and 80% for NaïveMCTS. The policy network uses a fixed time (around 3ms), and the remaining time is assigned to the tactical search.

Table 6 shows that both strategic algorithms greatly benefit from blending with a tactical algorithm. The gains are more substantial for the policy network, which now scores 56.7% against its *Puppet Search* counterpart. It also has a 4.3% higher overall win rate despite markedly poorer results against WorkerRush and AHTN-P. These seems to be due to a strategic mistake on the part of the policy network, which, if its cause can be detected and corrected, would lead to even higher performance.

5 Conclusions and Future Work

We have extended previous research that used CNNs to accurately evaluate RTS game states in small maps to larger map sizes usually used in commercial RTS games. The average win prediction accuracy at all game times is higher compared to smaller scenarios. This is probably the case because strategic decisions are more important than tactical decisions in larger maps, and strategic development is easier to quantify by the network. Although the network is several orders of magnitude slower than competing simpler evaluation functions, its accuracy makes it more effective. When the *Puppet Search* high-level adversarial search algorithm uses the CNN, its performance is better than when using simpler but faster functions.

We also trained a policy network to predict the outcome of *Puppet Search*. The win rate of the resulting network is similar to that of the original search, with some exceptions against specific opponents. However, while slightly weaker in playing strength, a feed-forward network pass is much faster. This speed increase created the opportunity for using the saved time to fix the shortcomings introduced by high-level abstractions. A tactical search algorithm can micro-manage units in contact with the enemy, while the policy chosen by the network handles routine tasks (mining, marching units toward the opponent) and strategic tasks (training new units). The resulting agent was shown to be stronger than the policy network alone in all tested scenarios, but can only partially compensate for the network's weaknesses against specific opponents.

Looking into the future, we recognize that most tactical search algorithms, like the MCTS variant we used, have the drawback of requiring a forward model of the game. Using machine learning techniques to make tactical decisions would eliminate this requirement. However, this has proved to be a difficult goal, as previous attempts by other researchers have had limited success on simple scenarios (Usunier et al. 2017; Synnaeve and Bessière 2016). Recent research avenues based on integrating concepts such as communication (Sukhbaatar, Szlam, and Fergus 2016), unit grouping and bidirectional recurrent neural networks (Peng et al. 2017) suggest that strong tactical networks might soon be available.

The network architecture presented in this paper, being fully convolutional, can be used on maps of any (reasonable) size without increasing its number of parameters. Hence, future research could include assessing the speed-up obtained by taking advantage of *transfer learning* from smaller maps to larger ones. Also of interest would be to determine whether different map sizes can be mixed within a training set. It would also be interesting to investigate the performance of the networks on maps that have not previously been seen during training .

Because the policy network exhibits some weaknesses against specific opponents, further experiments should be performed to establish whether this is due to a lack of appropriate game state samples in the training data or other reasons. A related issue is our reliance on labelled training data, which could be resolved by using reinforcement learning techniques, such as DQN (deep Q network) learning. However, full RTS games are difficult for these techniques, mainly because the only available reward is the outcome of the game. In addition, action choices near the endgame (close to the reward), have very little impact on the outcome of the game, while early ones (when there is no reward), matter most. There are several strategies available that could help overcome these issues, such as curriculum learning (Bengio et al. 2009), reward shaping (Devlin, Kudenko, and Grześ 2011), or implementing double DQN learning (Hasselt, Guez, and Silver 2016). These strategies have proved useful on adversarial games, games with sparse rewards, or temporally extended planning problems respectively.

References

Barriga, N. A.; Stanescu, M.; and Buro, M. 2014. Building placement optimization in real-time strategy games. In *Workshop on Artificial Intelligence in Adversarial Real-Time Games, AIIDE.*

Barriga, N. A.; Stanescu, M.; and Buro, M. 2015. Puppet Search: Enhancing scripted behaviour by look-ahead search with applications to Real-Time Strategy games. In *Eleventh Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 9–15.

Barriga, N. A.; Stanescu, M.; and Buro, M. 2017a. Combining scripted behavior with game tree search for stronger, more robust game AI. In *Game AI Pro 3: Collected Wisdom of Game AI Professionals*. CRC Press. chapter 14.

Barriga, N. A.; Stanescu, M.; and Buro, M. 2017b. Game tree search based on non-deterministic action scripts in real-time strategy games. *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG).*

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48. ACM.

Churchill, D., and Buro, M. 2011. Build order optimization in StarCraft. In *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*, 14–19.

Churchill, D., and Buro, M. 2013. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *IEEE Conference on Computational Intelligence in Games (CIG)*, 1–8. IEEE.

Collobert, R., and Weston, J. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, 160–167. ACM.

Devlin, S.; Kudenko, D.; and Grześ, M. 2011. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems* 14(02):251–278.

Foerster, J.; Nardelli, N.; Farquhar, G.; Torr, P. H. S.; Kohli, P.; and Whiteson, S. 2017. Stabilising experience replay for deep Multi-Agent reinforcement learning. In *Thirty-fourth International Conference on Machine Learning*.

Hasselt, H. v.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2094–2100. AAAI Press.

Ji, S.; Xu, W.; Yang, M.; and Yu, K. 2013. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence* 35(1):221–231.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Ontañón, S., and Buro, M. 2015. Adversarial hierarchical-task network planning for complex real-time games. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*, 1652–1658.

Ontañón, S.; Mishra, K.; Sugandh, N.; and Ram, A. 2007. Case-based planning and execution for real-time strategy games. In *ICCBR '07*, 164–178. Berlin, Heidelberg: Springer-Verlag.

Ontañón, S. 2013. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In *AIIDE.*

Ontañón, S. 2016. Informed monte carlo tree search for real-time strategy games. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, 1–8. IEEE.

Ontañón, S. 2017. Combinatorial multi-armed bandits for real-time strategy games. *Journal of Artificial Intelligence Research* 58:665–702.

Peng, P.; Yuan, Q.; Wen, Y.; Yang, Y.; Tang, Z.; Long, H.; and Wang, J. 2017. Multiagent Bidirectionally-Coordinated nets for learning to play StarCraft combat games.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; and Riedmiller, M. 2014. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Stanescu, M.; Barriga, N. A.; and Buro, M. 2014. Hierarchical adversarial search applied to real-time strategy games. In *Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 66–72.

Stanescu, M.; Barriga, N. A.; and Buro, M. 2015. Using Lanchester attrition laws for combat prediction in StarCraft. In *Eleventh Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 86–92.

Stanescu, M.; Barriga, N. A.; Hess, A.; and Buro, M. 2016. Evaluating real-time strategy game states using convolutional neural networks. In *IEEE Conference on Computational Intelligence and Games (CIG)*.

Sukhbaatar, S.; Szlam, A.; and Fergus, R. 2016. Learning multiagent communication with backpropagation. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc. 2244–2252.

Synnaeve, G., and Bessière, P. 2011. A Bayesian model for plan recognition in RTS games applied to StarCraft. In AAAI., ed., *Proceedings of the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2011)*, Proceedings of AIIDE, 79–84.

Synnaeve, G., and Bessière, P. 2016. Multiscale Bayesian modeling for RTS games: An application to StarCraft AI. *IEEE Transactions on Computational intelligence and AI in Games* 8(4):338–350.

Uriarte, A., and Ontañón, S. 2014. Game-tree search over high-level game states in RTS games. In *Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'14, 73–79.

Usunier, N.; Synnaeve, G.; Lin, Z.; and Chintala, S. 2017. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. In *5th International Conference on Learning Representations*.