

Using Neural Networks for Strategy Selection in Real-Time Strategy Games

Thomas Randall¹, Peter Cowling¹, Roderick Baker¹ and Ping Jiang¹

Abstract. Video games continue to grow in importance as a platform for Artificial Intelligence (AI) research since they offer a rich virtual environment without the noise present in the real world. In this paper, a simulated ship combat game is used as an environment for evolving neural network controlled ship combat strategies. Domain knowledge is used as input to the Artificial Neural Networks (ANNs) through scripts that run in parallel and feed their decisions to the ANNs. The ANNs then interpret these scripts and decide what strategy to perform. The results are compared to ANNs that have no such knowledge and tested to see how well the ANNs generalise.

1 INTRODUCTION

Computer games are increasingly becoming a platform for academic Artificial Intelligence (AI) research as they offer virtual environments without the inherent noise present in the real world, supporting the prediction of Laird and van Lent [1] that computer games could become a killer application for AI research.

Artificial Neural Networks (ANNs), a technique that uses a computational analogy with the human brain, is an active area of research with a lot of work being carried out in computer games [2 – 9]. Research shows that not only can ANNs be used in games, they can be the focus of the gameplay itself, as in the freely available NEROⁱ combat simulation game [3, 6 - 8].

Evolutionary Algorithms (EAs) are also an active area of research with a lot of work carried out in games [2 – 14]. EAs use an analogy with biological evolutionary process. Spronck uses Genetic Algorithms (GAs) to evolve script selection in a process called Dynamic Scripting [13, 14]. Dynamic scripting uses a population of scripts to control agents in a Real-Time Strategy (RTS) game. The scripts are initially picked at random and after each encounter each script element is allocated a fitness based on the performance of evolved scripts which use that element. This fitness is used to alter the probability of the scripts being selected in the next encounter. Spronck also describes a model for creating reliable adaptive game intelligence for use in commercial videogames [15] which Dynamic Scripting adheres to.

One common use of EAs is to evolve ANNs. Stanley and co-workers created a technique called NeuroEvolution of Augmenting Topologies (NEAT) that evolved both network weights and network topology at the same time [3 – 8]. NEAT uses historical

markings of topology changes, which allows NEAT to use operators to crossover different ANN topologies. NEAT starts off with a simple network topology adding connections and nodes through mutations in a process called complexification. These mutations often reduce the fitness but NEAT protects new innovation through speciation. Networks are assessed to see how similar they are to each other; if similar, they are placed into the same species, otherwise a new species is created. Individual networks do not compete against each other in NEAT; instead, species compete against each other. NEAT is particularly well suited to problems where it is difficult to determine appropriate network topologies, such as in the NERO game [3, 6 - 8] mentioned previously.

Incremental evolution is a process of increasing the complexity of the problem as the evolutionary process progresses [16, 17]. This can be done in different ways from altering the fitness measure to altering the environment that evolution takes place in. The aim is to use domain knowledge to guide evolution in the right direction for problems that are too complicated or impossible to evolve solutions directly.

The use and acquisition of domain knowledge is an active area of research, in particular Louis uses a Case-Injected Genetic Algorithm (CIGAR) to learn how to play strategy games [9, 10]. CIGAR injects chromosomes which encode previously successful strategies in the current population of strategies. CIGAR effectively learns from experience: as it plays a game, it keeps track of good strategies which might be used for future problems. CIGAR can be viewed as an incremental evolution technique as it uses prior knowledge to drive the evolutionary process.

In this paper we are controlling units in the ship combat game called DEFSIM, written in C#.NET, that simulates ship combat using the rules of the commercial video game DEFCONⁱⁱ, by Introversion Softwareⁱⁱⁱ. We aim to evolve intelligent behaviours for individual units in an RTS game in order to create intelligent players for the game overall. We use domain knowledge of strategies as input to ANNs to merge or produce a strategy based upon the input strategies. Most related research directly evolves ANNs that receive input from the game state [3, 6 – 8] or evolves mechanisms to select hand-written scripts in a specific order to create dynamic players [13, 14]. In this paper we pre-trained ANNs to carry out specific tasks and used them as scripts. Hence we can use evolution to combine a mixture of scripts, ANNs, and other techniques that aid the creation of intelligent-looking unit behaviours.

DEFSIM simulates DEFCON ship combat, whilst allowing us complete control over the source code, allowing us to tweak features as required. This is much more straightforward than directly using the DEFCON code, although this may shortly

¹ School of Computing, Informatics and Media, Univ. of Bradford, Bradford, BD7 1DP, UK. Email: {t.w.g.randall, p.i.cowling, r.j.s.baker, p.jiang}@brad.ac.uk.

change with [23]. DEFSIM also allows us to run experiments thousands of times more quickly than is possible than most commercial games available as it removes features unnecessary for AI research such as graphics rendering. This speed advantage offers a suitable environment to train/evolve ANNs. DEFSIM also allows for the addition of features as needed and allows us to run simplified rules of the game. As games become increasingly complex environments offering realistic physics, graphics and interactions the presence of noise increases, increasing the difficulty of using machine learning approaches. The use of a simulator helps remove some of this noise allowing us to concentrate on research. Once solutions have been found they may then be imported into the original game.

Ship combat works in a probabilistic manner in DEFSIM; combat between two battleships yields a 50% chance for either ship to be successful in battle. However, if a player has a local firepower advantage against an opponent in a battle, then these odds are changed in favour of the opponent with more ships [18, 19]. Learning to locally outnumber the opponent is a challenge for machine learning because the units must be made to cooperate and coordinate with each other, taking into account the number of opponents and allies in the nearby vicinity and moving strategically.

Evolution of solutions with a complex fitness function can often be too slow or render it impossible to find a solution without some additional guidance and domain knowledge. We investigate evolution of ANNs to interpret different scripts running in parallel and produce a strategy based upon them.

In section 2, we will discuss our experimental design and the different terrains we will use for evolution. We also describe the behaviour of the scripted AI against which the ANN players are tested. In section 3 we present our results and discuss our findings. In section 4 we present conclusions and plans for future work.

2 EXPERIMENT DESIGN

We use a population of 20 ANNs to select from different strategies to control a fleet of battleships in a game of DEFSIM against a team of scripted opponents. In DEFSIM, each ship has a 10% chance of hitting the closest opponent that is in the ship's attack radius (set to the same distance as the view radius) per frame. As the ANNs have to learn to approach the opponent ships and the opponent ships start outside the view radius, we have made it so that the ANNs have complete information (i.e. they have prior information as to the location of the opponent ships).

In this work we use SharpNEAT [20 - 22], a C#.NET implementation of NEAT [3 - 8] to evolve both the topology and weights of a population of ANNs. SharpNEAT also adds another process that is not available in the original NEAT design called pruning [22]. Pruning simplifies network topologies if the performance of a species in NEAT stagnates for a given period (i.e. no improvement in the species best fitness), which in these experiments is 200 generations.

The ANNs are evaluated on five different terrains with different strategies needed to win. We use five terrains to get a range of environments as units in an RTS game would need to operate on multiple terrains that the developers had not created, especially if the game allows user created terrains. These terrains are shown in Figure 4 to Figure 8 where ships marked with an

'E' are enemy (scripted) ships, ships marked with an 'F' are friendly (ANN controlled) ships and flags marked with a 'T' are targets. In these games, targets represent areas of importance, i.e. opponent bases. Damage inflicted on targets is worth significantly more than damage inflicted on opponent ships. The score for a game is shown in Equation (1). In these experiments the scripted players are playing a defensive role and the ANNs are playing an offensive role.

$$F_1 = \alpha \sum E_i + \beta \sum T_j \quad (1)$$

Where α and β are constants set to 1 and 5 respectively. E_i is the probability of enemy ship i being alive and T_j is the probability of enemy target j being alive.

```

foreach ship in myShips
    target = mostVulnerableTarget()
    if distance(ship, target) <
        (viewDistance(target) / 2) then
        ship.moveTowards(target)
    else
        if isInTargetViewDistance(opponent,
            target) then
            attacker =
                getStrongestAttacker(target)
            moveShipBetweenTargetAndAttacker(
                ship, target, attacker)
        else
            circleShipAroundTarget(ship, target)
        end if
    end if
end foreach

function mostVulnerableTarget()
    vulnerableTarget = nil
    targetDanger = 0
    foreach target in myTargets
        maxDistance = viewDistance(target) * 4
        danger = 0
        foreach op in visibleOpponents(target)
            dist = distance(op, target)
            health = getHealth(op)
            danger = danger +
                max((maxDistance - dist) *
                    health / maxDistance, 0)
        end foreach
        danger = (danger /
            amountOfVisibleOpponents(target)) *
            getHealth(target)

        if danger > targetDanger or
            vulnerableTarget = nil then
            targetDanger = danger
            vulnerableTarget = target
        end if
    end foreach

    return mostVulnerableTarget
end function

```

Figure 1. Pseudocode of scripted ships.

All of the terrains in these experiments use the same scripted opponents that try to defend the targets whilst the ANNs try to do as much damage to the targets as possible. The idea is to simulate a generic game strategy where the ANN-controlled player tries to attack the scripted player's base which has targets representing different building/important points that the opponent needs to defend. The targets are incapable of fighting, thus

they do not inflict damage against the ANN controlled ships. The scripted player’s pseudocode is shown in Figure 1. Note that to defend a target a ship stands between the target and the attacker. This is because in DEFSIM, combat is performed automatically and the attacker attacks the closest opponent with a 10% chance of hitting. Also note that health represents the probability of the ship being alive in the range [0 1] and uses Equation (2) to calculate damage inflicted. As a ship receives damage, the amount of damage it inflicts in a fight is reduced. This implicitly encourages the ANNs to minimize damage received as the fitness measure used in Equation (1) will be small when the ANN controlled ships take heavy damage.

$$H_t = H_{t-1}(1 - PA) \quad (2)$$

Where H_t is the defending ship’s new health, H_{t-1} is the defending ship’s current health, P is the probability of the attacking ship hitting the defending ship (in these experiments 0.1), A is the attacking ship’s current health.

The ANNs are evolved to select/merge different strategies to maximize their fitness using Equation (1) as the fitness measure. In these experiments, four different strategies are used as input for the ANNs. These are:

1. Attack opponent ships.
2. Attack opponent targets.
3. Group together with friendly ships.
4. Have more life than the opponent.

These four strategies are performed by pre-evolved ANNs. We leave the evolution of these behaviours outside the scope of this paper, but note that the ANNs could be replaced by hand-coded equivalent scripts with no fundamental difference to our results. We refer to these strategies as “ANN scripts” in the subsequent text. Strategy 4 represents a complex strategy where the ANNs will try to maximise their health whilst minimizing the opponents’ health, which could include behaviours such as running away when the ANNs health is small, and pressing an advantage with superior firepower in a particular location. Each ship is controlled individually by an identical ANN. All the ANN scripts output what a ship should do in terms of vertical (up and down) and horizontal (left and right) movement. The ANNs that use domain knowledge (via ANN scripts) do not receive direct input about the world state, instead receiving “advice” from the scripted behaviours. The inputs and outputs to the script interpreting ANNs are shown in Table 1 and Figure 2. The ANN scripts output an action, i.e. the output of the scripts is the degree of up/down and left/right movement.

This work has parallels with Dynamic Scripting [13, 14] and CIGAR [9, 10] as it uses domain knowledge in the evolution. The work presented here is different to Dynamic Scripting in that ANNs are evolved to interpret the scripts whereas Dynamic Scripting optimizes selection and ordering of hand-written scripts. CIGAR extracts domain knowledge from playing a game or watching someone play a game to create cases which are injected at suitable points into the population of an evolutionary algorithm. Here, we use knowledge in the form of ANN scripts. All the scripts fed into a master NEAT network in parallel and the ANNs are evolved to relate the scripts’ actions to final ship actions.

The ANNs will be compared to ANNs evolved without domain knowledge. We do this to see how the domain knowledge works against ANNs evolved directly to play the game. The inputs and outputs for the ANNs evolved without any domain knowledge are shown in Table 2 and Figure 3. The sensors used

in ANNs without domain knowledge (also shown in Table 2 and Figure 3) use Equation (3) as their input signal as represented in Figure 9.

In these experiments, all units have complete information about ship locations. Ship/target density in each direction is measured using

$$F_3 = \sum \frac{D-d_i}{D} \quad (3)$$

Where D is the maximum possible distance in a given direction (Up/Down and Left/Right) and d_i is the distance from the ANN controlled unit to a unit i . The sum is over all other friendly/enemy ships.

The distances of the ships are summed, as in [6, 7]. Summing the inputs allows for scalability to an unknown number of ships.

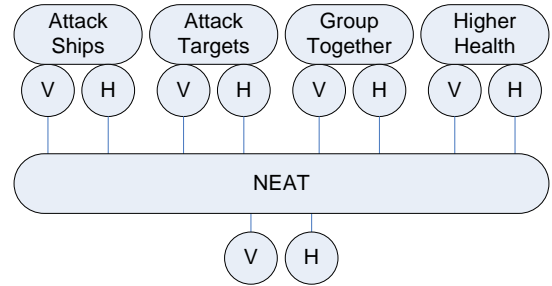


Figure 2. Knowledge base ANNs receive information from scripts. V represents vertical movements and H represents horizontal movements.

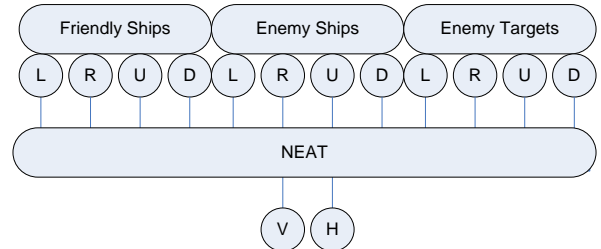


Figure 3. Game state ANN receives input about units in a game. L, R, U, D, represents the up, down left and right sensors. V and H represent vertical and horizontal movements.

Input	Description
0	Bias
1	Vertical movement from Script 1
2	Horizontal movement from Script 1
3	Vertical movement from Script 2
4	Horizontal movement from Script 2
5	Vertical movement from Script 3
6	Horizontal movement from Script 3
7	Vertical movement from Script 4
8	Horizontal movement from Script 4
Outputs	
0	Horizontal velocity
1	Vertical velocity

Table 1. List of Inputs and Output for the ANNs Receiving Input From Previously Trained ANNs

Input	Description
0	Bias
1	Left friendly ship Sensor
2	Up friendly ship Sensor
3	Right friendly ship Sensor
4	Down friendly ship Sensor
5	Left enemy ship Sensor
6	Up enemy ship Sensor
7	Right enemy ship Sensor
8	Down enemy ship Sensor
9	Left enemy target Sensor
10	Up enemy target Sensor
11	Right enemy target Sensor
12	Down enemy target Sensor
Outputs	
0	Horizontal velocity component
1	Vertical velocity component

Table 2. List of Inputs and Outputs for the ANNs Taking into Account Friendly Ships, Enemy Ships and Enemy Targets

Having a ship above and to the left of an ANN controlled ship will bring input to two sensors (the up and left sensors). It is possible for the ANNs to determine the direction of opponent ships when the inputs are compared. Two inputs are used for each direction to indicate density of hostile and friendly units in a particular direction. In both experiments, the networks are cloned so that each ship uses its own separate copy of the network. This is necessary since NEAT creates recurrent networks so each ship needs its own copy or recurrent input could potentially pollute the ANN “memory”. It also allows for scalability as different terrains have different amounts of ships in the game [3 – 8], and makes evolution easier.

In these experiments ANNs will be evolved against four of the five terrains shown in Figure 4 to Figure 8 playing 10 games on each of the four terrains per generation (in the sequence of 10 games on the first terrain then 10 on the next terrain and so on). The ANNs are also tested during evolution in the terrain they were not evolved against to see how well they generalise (with their progress on the test terrain not being considered during evolution). Both sets of experiments use Equation (1) as the fitness measure. The different experiments are:

- *Experiment One:* ANNs evolved using Figures 5, 6, 7, 8 and tested on Figure 4.
- *Experiment Two:* ANNs evolved using Figures 4, 6, 7, 8 and tested on Figure 5.
- *Experiment Three:* ANNs evolved using Figures 4, 5, 7, 8 and tested on Figure 6.
- *Experiment Four:* ANNs evolved using Figures 4, 5, 6, 8 and tested on Figure 7.
- *Experiment Five:* ANNs evolved using Figures 4, 5, 6, 7 and tested on Figure 8.

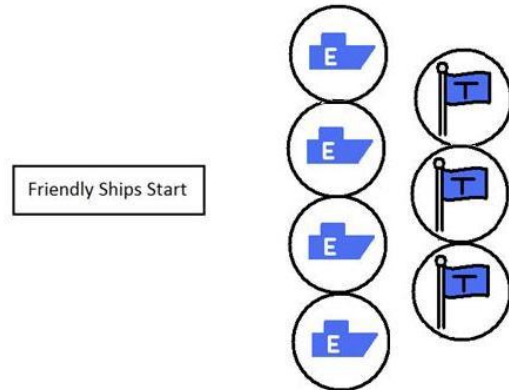


Figure 4. The ANN controlled ships start to the left with targets to the right defended by enemy ships.

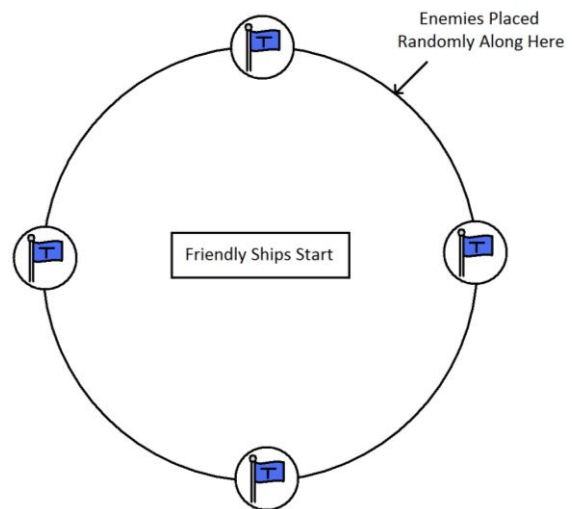


Figure 5. The ANN controlled ships start in random positions in the middle with four targets around them. The Enemy ships start in random positions on the circle that the targets sit on.

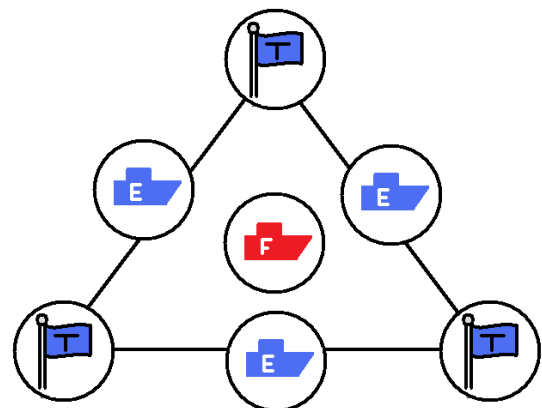


Figure 6. Three ANN controlled ships start in the centre (diagram only shows one) and three targets in a triangle around the ANN controlled ships. The enemy ships start on the lines between the targets.

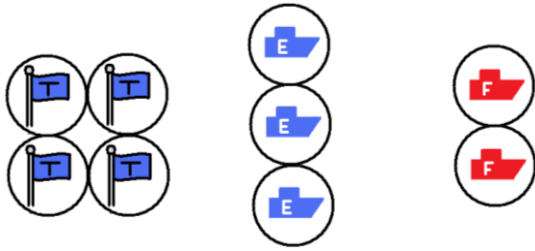


Figure 7. The targets are grouped together so that it is easier for the enemy ships to defend them.

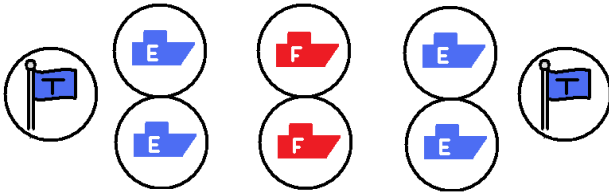


Figure 8. The ANN controlled ships start in the centre with targets at either side defended by enemy ships.

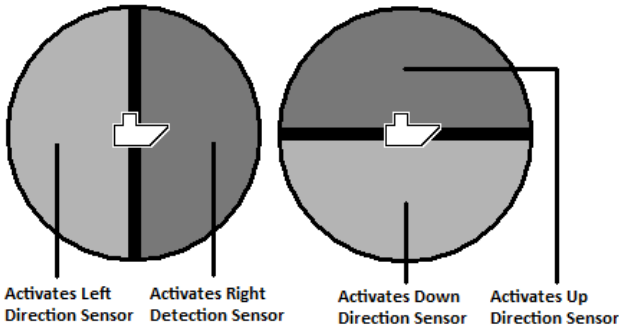


Figure 9. The zones that cause network inputs to the directional network. The circle around the ship is the ship's view radius.

3 RESULTS

Figure 10 to Figure 14 show the results of evolution for 500 generations, using Equation (1) as the fitness measure, for experiments one to five respectively. The default SharpNEAT parameters [20] were used for all experiments with the exception of the initial connection probability (the probability that a connection exists between two nodes for the initial starting population) which was set to 0.4 so that the initial population started with some connections rather than starting with few or no connections. As a small population (of 20) was used, many generations could occur with a lower initial connection probability before enough connections were made for the ANNs to perform well. All results show the 5th best genome of the population and are averaged over 10 runs of 10 games per evaluation with a population size of 20. Over a good deal of empirical investigation we have found the 5th best to be a robust statistic as it gives a good description of behaviours learnt whilst removing most of the chance inherent in experimental noise. We do not use the mean which includes 'lucky' and 'unlucky' outliers. Also note that the different terrains have different amounts of units and consequently the different results are not to the same scale as it

is possible to reach a higher fitness if there are more opponents (although not necessarily easier).

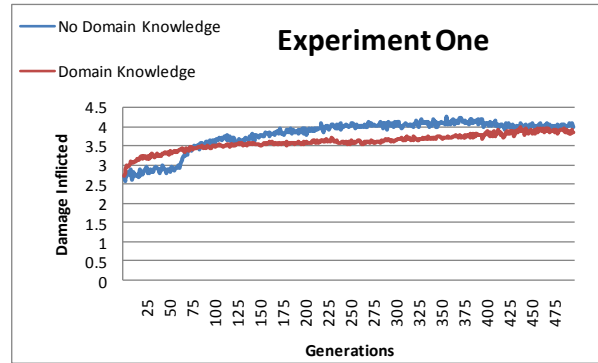


Figure 10. Experiment one evolved using Equation (1) and against Figure 5, Figure 6, Figure 7 & Figure 8.

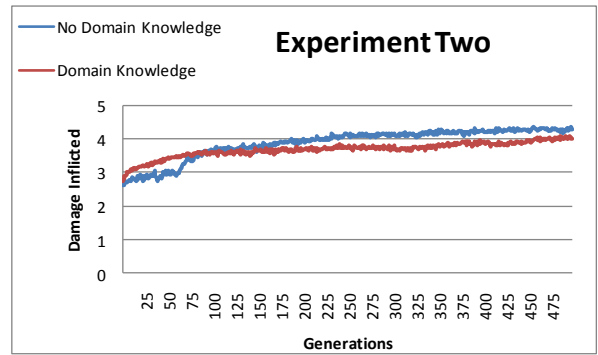


Figure 11. Experiment two evolved using Equation (1) and against Figure 4, Figure 6, Figure 7 & Figure 8.

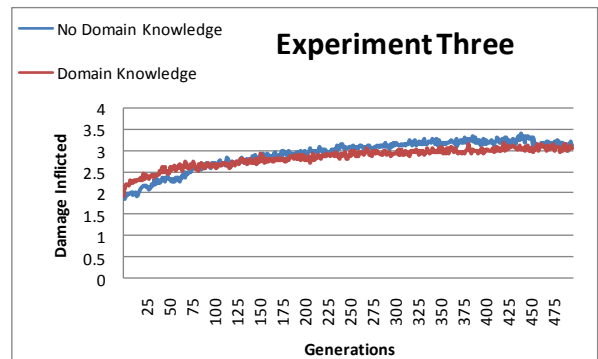


Figure 12. Experiment three evolved using Equation (1) and against Figure 4, Figure 5, Figure 7 & Figure 8.

It is clear from these results that generally ANNs evolved without domain knowledge, but have knowledge of the world state, produce a higher fitness than ANNs with domain knowledge (and no access to raw state information). However, ANNs that use domain knowledge evolve quicker and produce better initial results in early iterations. This happens for all experiments except experiment five (Figure 14) where both perform equally. ANNs with domain knowledge appear to evolve faster because they already have strategies that tell them how to play the game.

Once the ANNs with knowledge of the world state learns an initial strategy, they start producing good results that perform better than ANNs that use domain knowledge, presumably through optimisation of their strategy, something which is harder for the ANNs with domain knowledge because they do not have raw information about the game state and can only optimise the strategies from their knowledge base. We can also see that evolution had not finished at the end of the experiments. In all the experiments, it takes the method without domain knowledge a while to find an initial plausible strategy (so that domain knowledge has value when CPU time may be limited). Absence of a plausible initial strategy can often hinder evolution as is the reason for much research into incremental evolution [16, 17]. In order to perform well, the ANNs need to evolve a general purpose strategy that works across multiple terrains. This would also explain the sudden jump in fitness for ANNs that do not use domain knowledge in experiments 1-4 (Figure 10 to Figure 13) as the ANNs find an appropriate initial strategy.

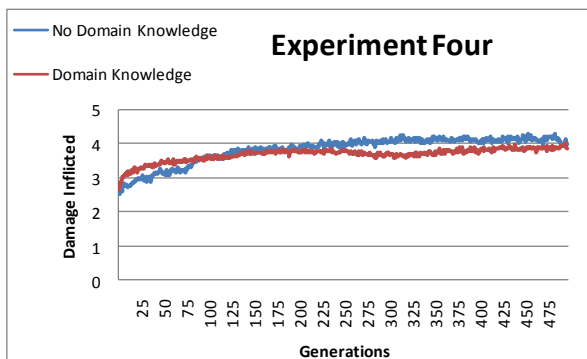


Figure 13. Experiment four evolved using Equation (1) and against Figure 4, Figure 5, Figure 6 & Figure 8.

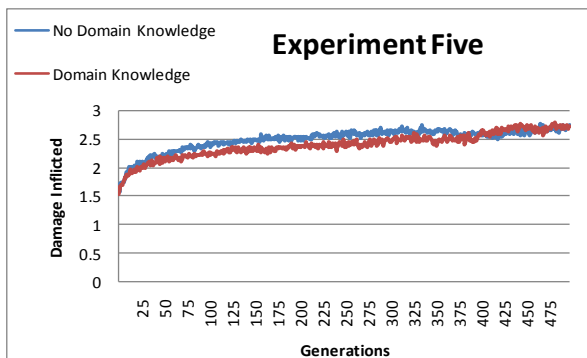


Figure 14. Experiment five evolved using Equation (1) and against Figure 4, Figure 5, Figure 6 & Figure 7.

During evolution the ANNs were also tested against an unseen terrain (although the results of this were not taken into account during evolution) to see how well they generalize on unseen test terrains. Results are shown in Figure 15 to Figure 19. The ANNs were not evolved on the unseen terrains, instead, they were tested during their evolution on the training terrains to investigate their performance changes.

Figure 15 shows the results for experiment one on the unseen terrain shown in Figure 4 using (1) as the fitness measure. It is clear from these results that once again ANNs that domain

knowledge starts off slightly better but is eventually overtaken by ANNs that do not use domain knowledge, but have more complete knowledge of the world state. Towards the end of the evolution cycle, the ANNs without domain knowledge start to decrease in fitness whilst ANNs with domain knowledge start to improve. This could be down to strategies learnt throughout evolution having a negative effect on this terrain as the scripted opponents defending the targets better against the strategies used. Note that whilst ANNs with domain knowledge start to decrease in fitness towards the end of evolution, they still perform significantly better than the initial starting fitness. As evolution is ended after 500 generations it is not clear if evolution of ANNs that use domain knowledge would have continued improving but it strengthens our previous hypothesis that if more time was given, the ANNs that use domain knowledge would start to outperform ANNs that do not.

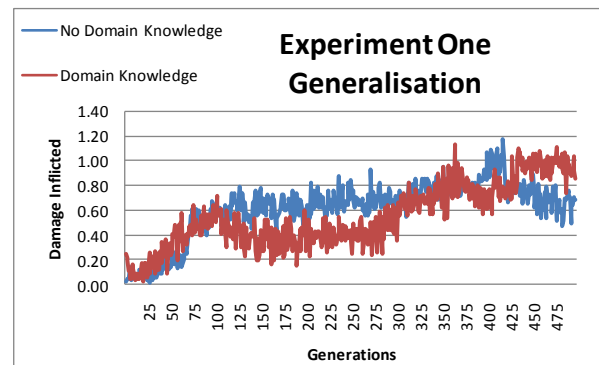


Figure 15. Experiment one generalizing on Figure 4 using Equation (1) as the fitness measure.

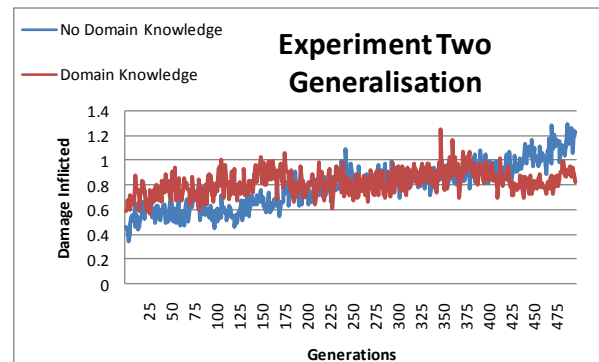


Figure 16. Experiment two generalizing on Figure 5 using Equation (1) as the fitness measure.

Figure 16 shows the results of experiment two's of generalization on the unseen terrain shown in Figure 5. It is interesting to note that once again the use of domain knowledge begins by producing better generalisation abilities although towards the end of the evolution cycle this tails off. ANNs that utilise domain knowledge does not really improve during the evolution cycle indicating that strategies evolved for training terrains do not really apply to the terrain shown in Figure 5. Figure 17 to Figure 19 show the remaining results for experiment three to five respectively at their ability to generalise. It is clear that the ability to generalise does not improve much during evolution for

both types of ANNs with the differences not being statistically significant. The lack of improvement could indicate that the strategies learnt during evolution do not apply to the unseen terrains. It could also indicate that the strategies learnt could potentially be terrain specific. The fact that ANNs without domain knowledge performs well on unseen terrains agrees with our previous hypothesis that the ANNs have learnt a general purpose strategy rather than a specific strategy for a terrain.

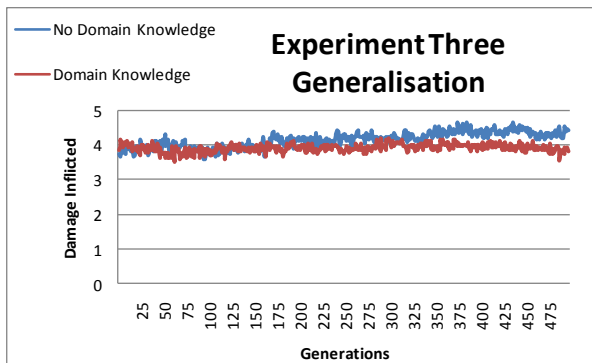


Figure 17. Experiment three generalizing on Figure 6 using Equation (1) as the fitness measure.

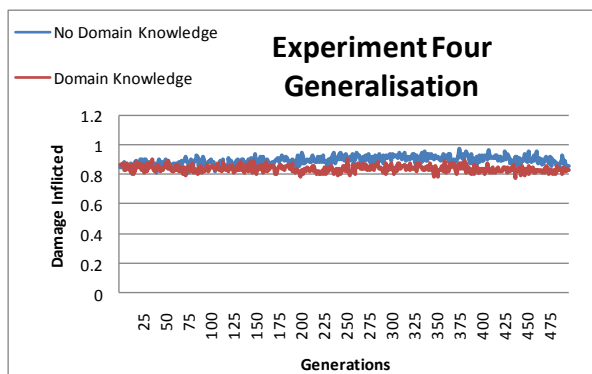


Figure 18. Experiment four generalizing on Figure 7 using Equation (1) as the fitness measure.

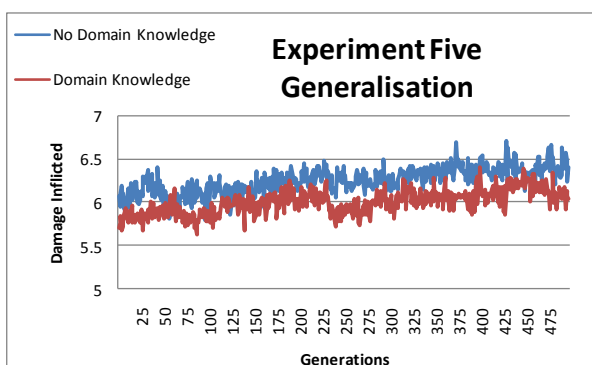


Figure 19. Experiment five generalizing on Figure 8 using Equation (1) as the fitness measure.

4 CONCLUSION & FURTHER WORK

We created DEFSIM, a C#.NET implementation of DEFCON in order to successfully perform the experiments presented here. This allows us to create many game scenarios for testing. We have shown that ANNs can be used to select/merge different scripted strategies to produce an effective player.

Whilst Dynamic Scripting can be used to select appropriate scripts for a given scenario, we have shown that ANNs can be used to “merge” different strategies that are running in parallel. In a sense, we have used an ANN in place of a Finite-State Machine, which is normally used to make a “hard” selection of which scripts to run, to make a “soft” selection via the network. The ANNs that used the scripts evolved good players quickly on terrains they were trained for, often in less than 60 generations. With a population of 20 like in these experiments, this does not take long to evolve. On a P4 1GHz processor, it took 1 CPU day to evolve 500 generations for one run of the experiment; the vast majority of this time was taken up by the ANNs playing DEF-SIM.

Generally, we observed that ANNs with domain knowledge perform better than ANNs without domain knowledge at the start of evolution, but after many iterations the players without domain knowledge (but with full state information) produce a higher fitness. The players with domain knowledge only had information from the ANN scripts and did not receive information about the game state. Another problem is that the ANNs could only ever come up with strategies that are in the knowledge base, in this case, they could only ever use one of the four strategies or a mixture of them. The ANN that does not use domain knowledge could potentially be using strategies that we, the writers, have not thought of. With all these limitation the domain knowledge based ANNs continued to evolve and, if more time was given, could potentially out perform ANNs without the knowledge, presumably through novel, unforeseen, combinations of these strategies.

For future work we intend to consider ANN players which have information about the game state as well as domain knowledge via ANN scripts to see if it aids the evolution process. We also intend to increase the rule base for the scripts selection to allow for more complex strategies. We also intend to improve the DEFSIM environment to allow new units and rules to further enrich the space of possible strategies.

ACKNOWLEDGEMENTS

This research was funded by the Engineering and Physical Science Research Council (EPSRC). We gratefully acknowledge the contribution of Introversion Software Ltd who donated the DEFCON source code for use in this work. We would also like to thank Stephen Remde and Colin Ward for proofreading.

REFERENCES

- [1] Laird, J.E., and van Lent, M. 2000. Human-level AI's Killer Application: Interactive Computer Games. *AAAI Fall Symposium Technical Report*, 80-97. North Falmouth, Massachusetts.
- [2] Bryant, B.D., and Miikkulainen, R. 2006. Evolving Stochastic Controller Networks for Intelligent Game Agents. *Proceedings of the*

- 2006 Congress on Evolutionary Computation (CEC 2006), 3752-3759.
- [3] D'Silva, T.; Janik, R.; Chrien, M.; Stanley, K. O.; and Miikkulainen, R. 2005. Retaining Learned Behavior During Real-Time Neuroevolution. *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005)*.
- [4] Stanley, K. O., and Miikkulainen, R. 2002. Efficient evolution of neural network topologies. In *Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02*, 1757-1762. CEC. IEEE Computer Society, Washington, DC.
- [5] Stanley, K. O. and Miikkulainen, R. 2004. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*.
- [6] Stanley, K. O.; Bryant, B.D.; and Miikkulainen, R. 2005. Evolving Neural Network Agents in the NERO Video Game. *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*, 182-189. Piscataway, NJ: IEEE Press.
- [7] Stanley, K. O.; Bryant, B.D.; and Miikkulainen, R. 2005. Real-time neuroevolution in the NERO video game. *IEEE Trans. Evolutionary Computation* 9(6): 653-668 (2005)
- [8] Stanley, K.O.; Bryant, B.D.; Karpov, I.; and Miikkulainen, R. 2006. Real-Time Evolution of Neural Networks in the NERO Video Game. *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, 1671-1674. Menlo Park, CA: AAAI Press.
- [9] Louis S.J., and Miles, C. 2005. Playing to learn: Case-injected genetic algorithms for learning to play computer games. *IEEE Transactions on Evolutionary Computation*. v9 i6. 2005, 669-681.
- [10] Louis S.J., and Miles, C. 2005. Combining Case-Based Memory with Genetic Algorithm Search for Competent Game AI. *ICCBW Workshops 2005*, 193-205.
- [11] Miles, C., and Louis S.J. 2006a. Co-evolving real-time strategy game playing influence map trees with genetic algorithms. In *Proceedings of the Congress on Evolutionary Computation*, Vancouver, Canada, 2006. IEEE.
- [12] Miles, C.; Quiroz, J.; Leigh, R.; and Louis, S.J. 2006b. Co-evolving influence map tree based strategy game players. In *Proceedings of the 2006 IEEE Symposium on Computational Intelligence in Games*, IEEE Press, 2007.
- [13] Spronck, P.; Sprinkhuizen-Kuyper, I.; and Postma, E. Enhancing the Performance of Dynamic Scripting in Computer Games, *Proceedings of the 4th International Conference on Entertainment Computing (ICEC 2004)*, 2004.
- [14] Spronck, P.; Ponsen, M.; Sprinkhuizen-Kuyper, I.; and Postma, E. 2006. Adaptive Game AI with Dynamic Scripting. *Machine Learning*, Vol. 63, No. 3, 217-248.
- [15] Spronck, P. 2005. A model for reliable adaptive game intelligence. In *IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games*, 95,100.
- [16] Christensen, A.L., and Dorigo, M. 2006. Incremental Evolution of Robot Controllers for a Highly Integrated Task. *Proceedings of The Ninth International Conference on the Simulation of Adaptive Behavior SAB'06*, 473-484. Rome, Italy.
- [17] Gomez, F., and Miikkulainen, R. 1997. Incremental evolution of complex general behavior. *Adapt. Behavior*. 5, 3-4 (Jan. 1997), 317-342.
- [18] Lanchester, F., 1916. *Aircraft in Warfare: the Dawn of the Fourth Arm*, Constable and Co. Ltd, London.
- [19] Pettit, L.I.; Wiper, M.P.; Young, K.D.S. 2003. Bayesian inference for some Lanchester combat laws. *European Journal of Operational Research*, Volume 148, Number 1, 152-165(14). 1 July 2003.
- [20] C. Green. *SharpNEAT homepage*. <http://sharpneat.sourceforge.net/>, 2003-2009.
- [21] D'Ambrosio, D.B.; Stanley, K.O. A Novel Generative Encoding for Exploiting Neural Network Sensor and Output Geometry. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*. New York, NY: ACM, 2007. 8 pages.
- [22] Lockett, A.J.; Chen, C.L.; Miikkulainen, R. Evolving explicit opponent models in game playing, *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, July 07-11, 2007, London, England
- [23] Robin Baumgarten. DEFCON API <http://www.doc.ic.ac.uk/~rb1006/projects:api>, 2009.

ⁱ Available at <http://www.nerogame.org/>

ⁱⁱ <http://www.everybody-dies.com/>

ⁱⁱⁱ <http://www.introversion.co.uk/>