

Improving Control Through Subsumption in the *EvoTanks* Domain

Tommy Thompson, Fraser Milne, Alastair Andrew & John Levine

Abstract—In this paper we further explore the potential of a decentralised controller architecture that places multi-layer perceptrons within a subsumption hierarchy. Previous research exploring this approach proved successful in generating agents that could solve problems while coping with new reactive stimuli. However there were many unresolved questions that we wished to explore. In this paper we explore the use of our architecture with iterative training, increased controller modularity and conflicting goals. Results provide some interesting insights into the potential this method could have to agent designers.

I. INTRODUCTION

Artificial Neural Networks (ANNs) combined with Evolutionary Algorithms (EAs) is a popular approach to agent design. Providing an automated approach to modularity where the designer need only create an effective learning process to train the network; learning how best to break down raw sensor data and turn it into intelligent action. However this approach is heavily constrained by the number of synapses (or weights) within the network, while each synapse provides an extra dimension of freedom for the search process to find potential solutions it also increases the complexity. Ultimately the designer must realise the point at which this balance is disturbed and look for further means of organisation and control. This can be found in a range of different agent architectures and staged training approaches.

Our previous work in [1] takes inspiration from subsumption architectures; task independent modules that interact and ‘subsume’ one another in order to achieve a desired behaviour [2], [3], [4], [5]. Our training approach was applied to our tested *EvoTanks*, with agents generating robust and capable ANNs for navigation, obstacle avoidance and combat behaviours. While these prove highly effective their full potential has yet to be fully explored.

In this paper we explore areas of potential improvement from [1]. Taking our original designs, we provide a series of experiments exploring results in increased modularity, iterative training mechanisms and combining de-coupled goal controllers.

The layout of this paper is as follows; we begin with a brief introduction of our *EvoTanks* domain in Section II, followed by a breakdown of the research previously conducted in the domain in Section III. We give a recap of our agent architecture in Section IV, with our research questions and experimentation found in Section V. We finish with a discussion of the results found in Section VI and our closing comments in Section VII.

Alastair Andrew, John Levine and Tommy Thompson are with the Strathclyde Planning Group, University of Strathclyde, Glasgow, G1 1XH, UK, email: forename.surname@cis.strath.ac.uk.

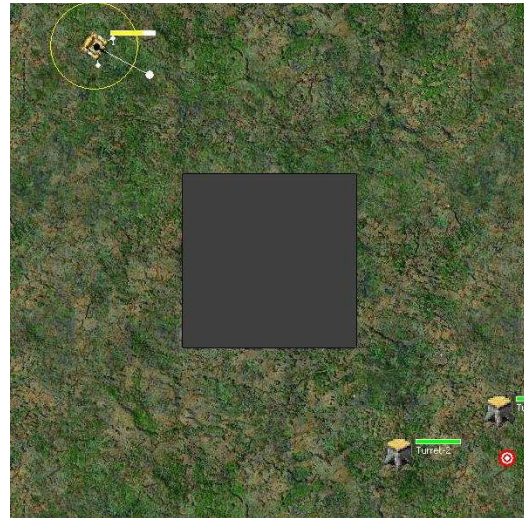


Fig. 1. An screenshot of the *EvoTanks* domain, in this sample problem, the agent is attempting to reach the waypoint in the bottom right corner, whilst avoiding the large obstacle in the centre of the arena as well as any incoming fire from the two turret agents placed around the waypoint.

II. THE *EvoTanks* DOMAIN

EvoTanks is designed with the intention of allowing agents to learn how to solve local tasks using EA approaches. The original purpose of the domain was to mimic *Combat* released for the Atari 2600 in October 1978. *EvoTanks* incorporates many of the features of *Combat*. Existing as a square arena measuring 600 by 600 pixels in size. Agent are limited to basic movement; forwards, backwards, turn left or right and firing of the cannon. Each agent is capable of activating one or more of these actions at each discrete time-step of the simulation. Movement is fixed to a predefined distance (2 pixels) or angle (4 deg) per update. Once any action is committed, there is a minimum delay before that same action can be committed again. The cannon is fixed in front of the tank and carries an unlimited supply of shells. Each agent will start with a maximum health of four points, and will suffer a loss of one point for each collision with the environment or enemy fire. Regardless of the intended goal, a tank is limited to 1000 time-steps in any evaluation.

III. RESEARCH IN *EvoTanks*

Due to space constraints, we do not wish to recap relevant research pertaining to our endeavours. Instead we focus on work conducted to date to give the reader understanding of our overall direction. Sufficient reading detailing related work and technical background can be found in [1] and to a lesser extent in [6].

Our preliminary research found in [7] was to explore the feasibility of evolving ANN driven agents in *EvoTanks*. Our initial findings proved positive; providing agents that could learn to counteract features of non-player character (NPC) behaviour. However these agents were prone to trappings of local maxima within the fitness space. Hence further research carried out in [6] explored different ways to achieve strong, general behaviours. This was achieved through the use of a 1+1 evolutionary strategy evaluating against a variety of scripted agents, as well as the use of a competitive co-evolution learning model.

The focus of research in this paper is driven directly from that chronicled in [1]. In this paper we explore the use of top-down subsumption to build a layered architecture utilising our ANN controllers. Taking inspiration from the early research in this area by Brooks in [2], [3] as well as the only other known research published in this vein by Togelius in [8]. We formulated our agent architecture (explored in Section IV) and tested it against a variety of problems. These problems often required controllers with two or three ANNs placed strategically within the subsumption framework. For example, the sample problem shown in Figure 1 necessitates a 3-layer controller for navigation, obstacle avoidance and dodging incoming enemy fire.

The final results in [1] show that our approach was effective in compensating for added complexity in the environment and provides a ‘plug n play’ approach to behaviour construction. However we were left with many unresolved questions and interests that we wished to pursue as detailed in Section V.

IV. AGENT DESIGN

In this section we provide background on the design of our architecture and the controllers that we utilise. Should the reader desire further clarification and detail, please refer to the related work section in [1].

A. Controller Design

Each layer of the subsumption hierarchy is composed of a feed-forward, multi-layer ANN. Each ANN is designed with four layers; input, output and two intermediate layers of hidden neurons. In most instances the hidden neurons are kept within a range of three to five neurons per layer. Each hidden neuron utilises a TanH transfer function to reduce the initial bias of the network. We often constrain the size of the network with typically less than 30 synapses per controller. This allows for faster training times as shown in [1]. An example controller composed of three networks is shown in Figure 2.

A controller will typically use two to four inputs from the domain. These inputs are normalised within ± 1 , with the intention of smoothing and restricting the potential input space. The actual inputs selected are dictated by the goals of the controller and are covered later in this section.

Each network is able to provide up to n outputs in the final layer, where n is a subset of the *EvoTanks* actions; forward/backward movement, left/right rotation and the firing

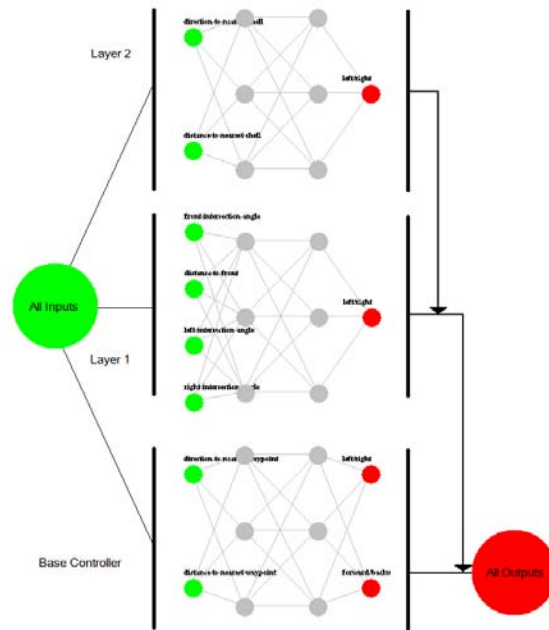


Fig. 2. An example Subsumption Neural Architecture (SNA), where three individual controllers are placed atop one another to dictate a hierarchy of execution. Note that the agents will utilise a subset of all available inputs and activate a subset of the outputs.

of the cannon. In the event of positive output, the first neuron will trigger the ‘forward’ action, the second will trigger the ‘turn right’ action and the latter will fire the cannon provided the output value exceeds ≈ 0.01 . Negative output will generate the opposite effect in all instances. For the movement and rotation outputs there exists a small range that operate as ‘null’ outputs, resulting in no-op. The weight of each synapse is constrained within a range of ± 5 , hence each chromosome is a series of weight values within this range.

Each controller was designed as either a *base controller*, where the controller has some goal driven objective that drives its performance, or as a *layer controller* which is designed to operate in conjunction with the base to assist in dealing with information de-coupled from the goal. Each layer controller is designed to associate with a feature that can be added to a problem we are training against. Typically layer controllers seldom activate until they are required to assist with the issues pertaining to their function (for example, a obstacle avoidance controller only ever activates once the sensors recognise there are obstacles within the vicinity of the agent).

Action selection for the controller is dictated across each of the available outputs that our agent can commit. At each time-step, all networks within the hierarchy process outputs based on the current available data. Once completed, the top-down subsumption approach is utilised across all controllers to select the committed action. Agent movement, rotation and firing are triggered by the network which has committed this output that holds the highest position in the hierarchy. This

allows more than one controller to be in control of the overall agent behaviour if necessary. A suitable example of this is navigation, where a high level controller may wish to direct an agent away from nearby obstacles (thus restricting the rotation output) while allowing the agent to continue toward its objective by *not* restricting the movement output.

B. Learning Methodology

Our original training methodology is laid out in detail in this section and summarised in Algorithm 1.

Algorithm 1: A breakdown of the training process for individual subcontrollers.

Input: A list of controllers Q of size l , where the tail is the top controller in the hierarchy and vice versa.

Result: A trained hierarchy of ANNs tailored to the subsumption paradigm, R

Assign fitness function for training from Q_0

addController(Q_0, R)

for $i \leftarrow 0$ to l do

 for $j \leftarrow 0$ to Eval do

 | best $_i \leftarrow \text{train}(Q_i)$

 end

 freeze(best $_i, Q_i$)

 if $i < (l - 1)$ then

 | addFeatureToEnvironment(Q_{i+1})

 | addController(Q_{i+1}, R)

 end

end

Given a series of ANNs, we begin by training the base network, which naturally is situated at the bottom of the subsumption hierarchy. From the base controller we formulate the problem instance for training. For a specified number of evaluations the controller is then trained against random instantiations of the problem. Once the specified number of evaluations is reached, then training concludes on this particular controller and is now ‘frozen’ in place, i.e. it will not be trained any further and the behaviour is now fixed. We take the next layer controller in the hierarchy, add it on top of the current architecture in place and train it. Now the learning process commences once again under the same fitness criterion, however the problem instance will be modified in order to reflect the changes that necessitate the current controller being used. For example, if we are about to begin training a controller designed for obstacle avoidance then obstacles are added into the environment. This cyclic process is repeated until all controllers have been trained.

The learning process utilised in this paper is an evolutionary strategies; a search and optimisation method from the family of evolutionary algorithms. Learning is conducted using the simple, yet aggressive 1+1 strategy; a standard random mutation hillclimber. In our original research we also conduct experiments using a $(\mu + \lambda)$ evolutionary algorithm. However for the purposes of presenting our research we resort solely to the 1+1 strategy as means to assess the

validity of our controller designs. In Section V, we deploy this method to assess and compare our modified training examples. The results of previous experiments in [6] indicate that the evolutionary strategy provides an upper-bound on the fitness that can be achieved within the defined landscape.

V. EXPERIMENTATION

In this section we raise and explore our research questions and highlight interesting features from each experiment that was undertaken. Unless otherwise stated, each experiment utilises the 1+1 evolutionary strategy across 250,000 evaluations. This will typically be spread across 5000 mutations with 50 games to assess each candidate. When training across multiple controllers these mutations are distributed evenly across each phase.

In this paper we seek to answer the following questions:

- Given that our original learning approach only allowed one learning phase per sub-controller. Would an iterative re-training allow for controllers lower in the architecture exploit controllers above it to improve performance?
- While our controllers were relatively small and fast to train, if we broke them apart using subsumption would we see any improvement?
- Lastly, can we take the goal-centric (base) controllers and use the subsumption to generate agents capable of dealing with conflicting goals?

A. Iterative Training

First we investigate whether modifications made to the learning process described in Section IV will result in improvement in resulting behaviours. One of our observations following the experiments conducted in [1] was the learning process de-coupled any controller from those above it in the hierarchy, whereas it was dependant on anything beneath it. Hence we considered whether we could modify the learning process to enforce a relationship with higher controllers.

Since a controller must be left sufficient freedom to explore the search space, any premature attempts at this stage to control its performance would result in failure to generate effective control, which was the reason we employed this learning process in the first place. As a result we turned to an iterative process of training; by re-training layers once those above them have been completed. If we take for example a 2-layer controller designed to navigate waypoints on the bottom layer and another to avoid nearby obstacles on the top. When the learning process takes place, we train to learn how to visit some given point in space. Once this is completed we then train to navigate around obstacles in the arena, all the while knowing it can reach the waypoint provided the top controller learns effective means of avoidance. The top controller will learn what is sufficient to allow that beneath it to operate effectively. What this experiment seeks to discover is if we return to the base controller, will we see any difference in

resulting behaviour and fitness as a result of re-training the navigation controller given that it can now overcome nearby obstacles?

The task shown in Figure 1 was the focus of our experiment; with the agent charged with navigating across the world while avoiding obstacles and incoming fire from our scripted *Turret* agent. In the original version, we train all three layers; *visit waypoint*, *detect obstacles* and *dodge shells* in sequence for an equal number of evaluations. In the modified version, we train through as usual, however once we reach the end of the *dodge shells* phase, we return back to the *visit waypoint* phase while retaining our current best controllers for all higher layers. Given that we have all controllers available we continue to train on the complete task instead of returning to the incremental modification of the environment. Each experiment is ran multiple times to ensure accurate qualitative measurements are made.

Table I provides statistics from 10 runs of each experiment on our 1+1 strategy. Furthermore we provide graphs of typical learning trends and the matching statistics from both the 1-iteration and 2-iteration experiments in Figures 3 and 4 respectively. Each graph presents the current best fitness of the evolutionary strategy as well as the average fitness (with standard deviation) of mutations as learning progresses. The statistical results provide some interesting findings, notably that on average the use of a further training iteration for each controller results in a greater final fitness. Furthermore the best fitness from the 2-iteration run surpasses any of our previous attempts, with a final fitness breaking the 0.7 barrier. We also see some interesting results in the learning trends. In Figure 4 we see our agent learn the first two controllers effectively despite the shorter training period. However it fails to achieve the same fitness as shown in Figure 3. What is encouraging to see is that the re-training actually improves this score throughout the next two phases. Sadly there is a small dip in fitness once it attempts to re-train the final controller. At this point it is worth considering whether a memory function should be considered to allow us to return to the previous version of a particular controller should the resulting fitness decrease. What is also interesting to observe is the standard deviation across mutations tightens during the learning process. Reducing the gap between the average and the best found solution far more than in the standard method.

If we compare the resulting behaviours from each of these results, we see some minor improvement through the second training method. While the controller continues to behave in the same manner; moving towards it's ultimate goal, moving around nearby obstacles in a 'wall hugging' manner and shifting direction in the presence of enemy behaviour. There appears to be a greater consistency and robustness to the controller as it appears to make less incorrect decisions.

B. Increasing the Modularity of Controllers

In this experiment we seek to explore whether increasing the modularity of controllers improves performance. This returns to a question raised during our experiments in [1] that perhaps removing or breaking apart the control of the

TABLE I

STATISTICS FROM 10 RUNS OF OUR 3 LAYER PROBLEM INSTANCES. WE PROVIDE RELEVANT STATISTICS FROM THE LEARNING PROCESS OF THE 1-ITERATION AND 2-ITERATION VERSIONS. RESULTS SUGGEST THAT THE ADDED PHASE OF TRAINING IMPROVES THE OVERALL PERFORMANCE.

| Statistic | 1 Iteration | 2 Iterations |
|------------------|-------------|--------------|
| <i>Maximum</i> | 0.6965 | 0.7158 |
| <i>Median</i> | 0.667 | 0.6446 |
| <i>Mean</i> | 0.571 | 0.6041 |
| <i>Std. Dev.</i> | 0.118 | 0.157 |

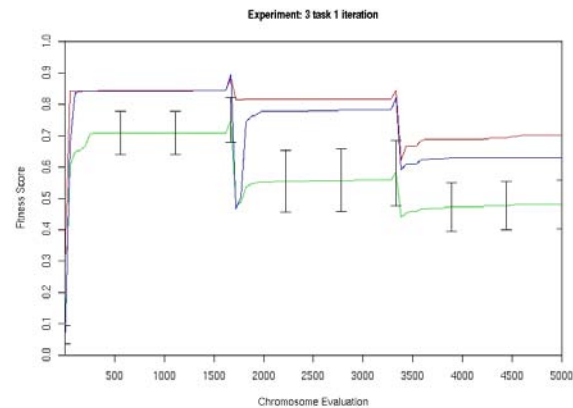


Fig. 3. This graph chronicles the best fitness and average mutation fitness with standard deviation from one of our 3-task experiment runs presented in Table I using only one round of training on the environment shown in Figure 1. These results are akin to that which we have shown previously in [1], with overall fitness suffering from minor reductions as time progresses due to the inclusion of new challenges in the environment. This however does not prevent the agent from completing the task prescribed.

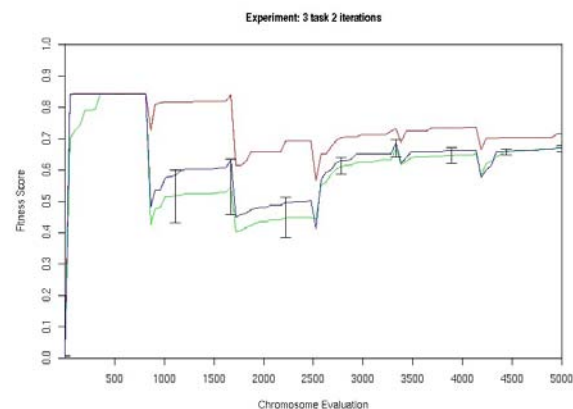


Fig. 4. This graph chronicles the learning trend from one of our 3-task experiment runs presented in Table I using two rounds of training. These results show an interesting trend as the agent learns the first two layers competently while struggling with the last layer. However when the second round of training occurs as all layers are re-trained we begin to see improvement prior to a minor yet disappointing dip in fitness once we re-train the top layer.

TABLE II

STATISTICS FROM 10 RUNS OF EACH OF OUR MODULAR EXPERIMENTS USING THE VISIT WAYPOINT CONTROLLER. WHILE THE DIFFERENCE IN MAXIMUM FITNESS VALUES IS MARGINAL, THE REMAINING STATISTICS SHINE AN INTERESTING LIGHT ON THE EFFECT THESE APPROACHES HAVE ON THE LEARNING PROCESS. INTERESTING TO NOTE THAT WHILE THE STANDARD DEVIATION ON AVERAGE IS RELATIVELY HIGH IN THE SUBSUMPTION EXPERIMENTS, THE FINAL RESULTS SHOWN IN FIGURE 5 INDICATE THAT THE VARIANCE REDUCES AS THE TRAINING CONCLUDES.

| Statistic | Standard | 2-Layer | 2-Layer Iterative |
|-----------|----------|---------|-------------------|
| Maximum | 0.8437 | 0.8441 | 0.8442 |
| Median | 0.8421 | 0.23511 | 0.8405 |
| Mean | 0.6442 | 0.5093 | 0.6524 |
| Std. Dev. | 0.05983 | 0.325 | 0.2862 |

agent may result in improvement. Our earliest observations to suggest this was that many of our *visit waypoint* controllers maintain forward movement and then reduce the difference in angle between the front of the tank and the waypoint. This led us to consider whether removing the distance input would generate different results, however it led to a poorer fitness. We also took similar interest in our *destroy target* controller, given that it relies on input that expresses the difference in angle between the enemy agent and the player. While this input has proved beneficial in the past it often appears that our controller is strongly dependant upon it. We were keen to see whether modifying the structure of this controller would result in different behaviour.

We begin by looking at the *visit waypoint* controller. For this process we undertake three experiments; a standard waypoint navigation task using our usual controller (for comparison purposes), a 2-layer subsumption controller where the base neural network utilises only the angle input, with the distance input on the top layer. The third experiment uses the same controller configuration however an iterative training process is employed.

Having conducted each experiment ten times, statistics are provided in Table II and graphs representing typical learning trends in Figure 5. The difference in final maximum fitness could be deemed marginal with the 2-layer approach being only slightly better on average. The remaining statistics however tell a different tale, notably the effect the subsumption has on the variance throughout training. In both subsumption examples we see large standard deviations, this could be attributed to the larger search spaces that the architecture permits. Despite poor initial results, by applying our iterative training the results drastically improve and the mutation variance drastically reduces in comparison with the traditional approach. Finally, analysis of the behaviours show virtually identical performance, with no real difference in the final product.

Next we explore the same process against the *destroy target* controller by pulling the ‘direction from enemy’ input out of the setup. We have three configurations; the standard *destroy target* controller utilising all three inputs, a single

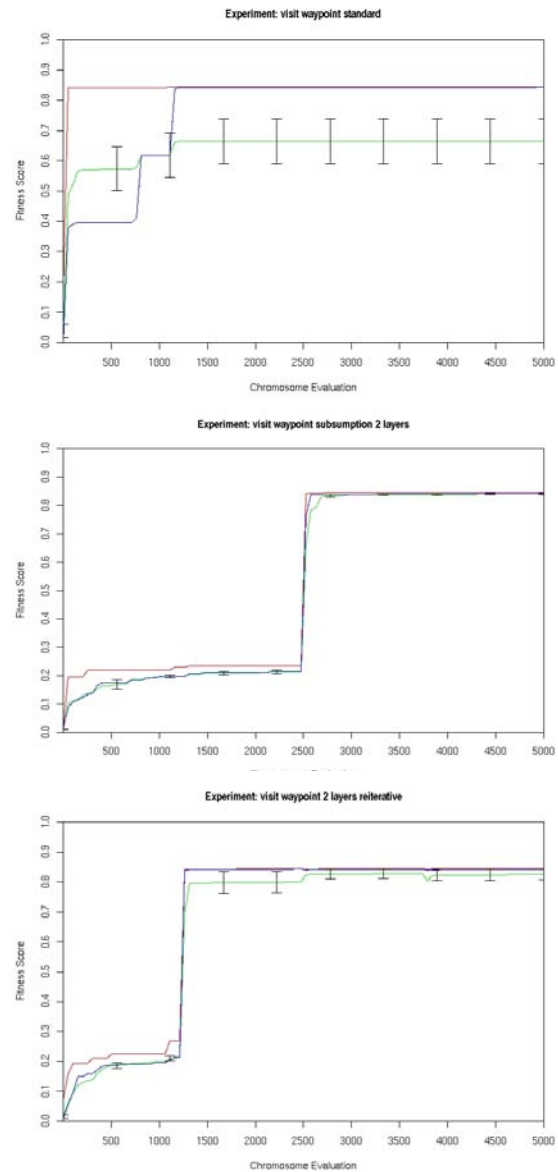


Fig. 5. Graphs of best fitness and average mutation fitness from our visit waypoint runs presented in Table II. While we see different trends throughout the learning process, the final results are very similar.

layer controller without the direction from enemy input and a 2-layer controller that moves the direction input into a higher layer. In this instance our agent is trained against all six of our scripted agents with strategies ranging from docile targets (Sitting Duck) to aggressive (Hunter) and defensive (Sniper).

Once again we conduct the experiment 10 times and provide statistics of these runs in Table III. We also provide typical learning trends in Figure 6. Firstly we note that removing the third input from a single layer controller results in better performance and secondly we see that the subsumption garners the best results both in terms of maximum and mean fitness. There is a small improvement in the best fitness found, this is interesting given the nature of the fitness function and the penalty for time taken. The

TABLE III

STATISTICS FROM 10 RUNS OF EACH OF OUR MODULAR EXPERIMENTS USING THE DESTROY TARGET CONTROLLER.

| Statistic | Standard (3-Input) | 2-Input | 3-Input 2-Layer |
|------------------|--------------------|---------|-----------------|
| <i>Maximum</i> | 0.834 | 0.8471 | 0.8529 |
| <i>Median</i> | 0.7425 | 0.8275 | 0.8261 |
| <i>Mean</i> | 0.68 | 0.807 | 0.8142 |
| <i>Std. Dev.</i> | 0.0667 | 0.063 | 0.0562 |

agent has become more efficient at eliminating the variety of enemies and surpasses the best results we have accrued to date. The learning patterns also show that by removing the third input not only is training marginally faster, but also results in greater stability in the mutations. With the mean fitness far closer to the best found in the learning process. Even more interesting that there is a minor improvement by adding the third input into a subsumption layer, we had hypothesised that the controller would simply ignore this higher layer. However it appears to have taken it in board and utilised it more effectively than in our previous designs. In terms of behaviour, the subsumption controlled agents are highly ruthless, taking on a behaviour that will chase an agent down to within a certain distance, at which point it will maintain that distance and continue firing.

C. Layered Base Controllers

Our final experiment explores the feasibility of utilising two base controllers in the same subsumption hierarchy. In our previous work we designed our layer controllers to provide extra functionality beyond the original capabilities of the base controllers. Given that the layer controllers only needed to react to local stimuli, this would allow the base controllers to continue as normal when the layer controllers did not need to interfere.

Our proposal for this experiment is to create a subsumption hierarchy using two of the goal-driven base controllers; *visit waypoint* and *destroy target*. The interesting challenge here is that the two controllers involved have conflicting goals. While one is interested in navigating to a static point, the other is tasked with combat and manoeuvres. We were interested in testing whether it would be possible to suppress the higher layers of a network despite them being fed with constant input.

To explore this concept, we have an agent compete against a pair of scripted tanks. The goal is to be the first agent to visit twelve randomly instantiated waypoints in the world. However either team will lose if a member of the opposing team is two waypoints ahead, furthermore should the enemy team be eliminated then this alleviates the two-waypoint restriction. Each agent is initially positioned in a race-grid style position facing the first waypoint. This problem is ideal since it necessitates navigating to each point as well as possible elimination of enemy agents. The 2-waypoint distance restriction enforces the need to continue moving and prevents the agent from simply charging aggressively at the enemy team.

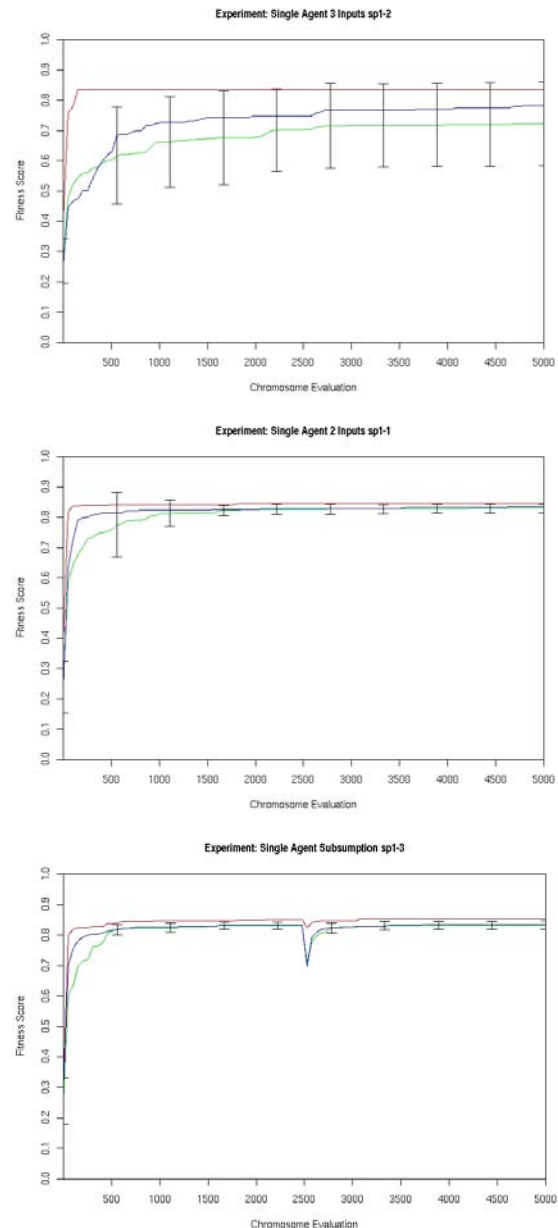


Fig. 6. Graphs representing best fitness and average fitness of mutations (inc. std. dev.) from our destroy target runs presented in Table III. Once again despite different trends throughout the learning process, the final results are very similar.

The experiment is conducted in three variations; the first utilises a neural network training against the visit waypoint task. Next is a 2-layer subsumption with a visit waypoint controller on the bottom and a destroy target controller *without* the cannon enabled. Lastly we run with a 2-layer subsumption using both controllers and an active cannon. The first experiment is used to compare how effective the agent is when only visiting waypoints. The second and third experiment provide the location of the nearest enemy agent, the cannon is only permitted in the latter as we assess whether having this information can be used for navigation

TABLE IV

STATISTICS FROM 10 RUNS OF EACH OF OUR WAYPOINT RACE EXPERIMENTS. USING A RANGE OF DIFFERENT CONTROLLERS THAT RELY ON BOTH VISIT WAYPOINT AND DESTROY TARGET CONTROLLERS. WHILE WE SEE IT IS SUBSTANTIALLY MORE TAXING TO UTILISE THE EXTRA GOAL CONTROLLER, RESULTS INDICATE IT IS POSSIBLE TO GENERATE EFFECTIVE SOLUTIONS.

| Statistic | 1-Layer | 2-Layer No Cannon | 2-Layer w/Cannon |
|-----------|---------|-------------------|------------------|
| Maximum | 0.678 | 0.5269 | 0.6640 |
| Median | 0.657 | 0.3883 | 0.4178 |
| Mean | 0.6532 | 0.3615 | 0.4480 |
| Std. Dev. | 0.0133 | 0.1266 | 0.1323 |

purposes.

Results from this experiment can be seen in Table IV as well as graphs of the best runs of these experiments in Figure 7. It is important to note that in all three experiments, we permit a short period of time to learn the visit waypoint controller before entering enemy agents into the playing field. Once completed, agents will start from one of three ‘grid’ positions at random. This explains the initial boost in fitness followed by the sharp drop-off after the first 500 chromosome evaluations. Given the difficulty of this challenge it was not expected that our agents could succeed to as great an effect as before, nonetheless they appear to have performed admirably in the first and last experiment. Using only one controller we see that there is still a significant challenge for the agent to overcome, this is noticeable in the top graph of Figure 7, where we see the sharp dip in fitness once the racing commences, nonetheless the agent recovers. In our second experiment, the agent fails to perform as well, struggling to succeed on average. However when we provide the ability to eliminate the enemy agent we see a drastic improvement, the agent is successful in clearing the race almost as effectively as without. Despite a poor average performance, there is clearly improvement over the second approach. The resulting behaviours are interesting to observe, the single network is the most efficient in terms of racing. The agent is more effective at manoeuvring and will often reach the waypoint first when in a dead heat with an enemy agent. When we supply the enemy information for navigation, we see the agent will often follow the enemy players as closely the possible, running on the assumption it will also reach the desired waypoint. However this is not always the case, as the agent will sometimes miss the desired waypoint, as a result it will either ignore it and continue chasing tanks or it will double back to visit the waypoint. Both of these behaviours will often result in failure. The aggressive racer provides an interesting controller to observe, it retains much of the ‘chase’ behaviour from before, however it ensures it will visit the waypoints and seldom misses any. Furthermore when the agent is close behind the enemy it will fire shells with the intention of eliminating the enemy. We often see instances where our agent will be the only racer to reach the finish line, as it will eliminate the enemy players during the race.

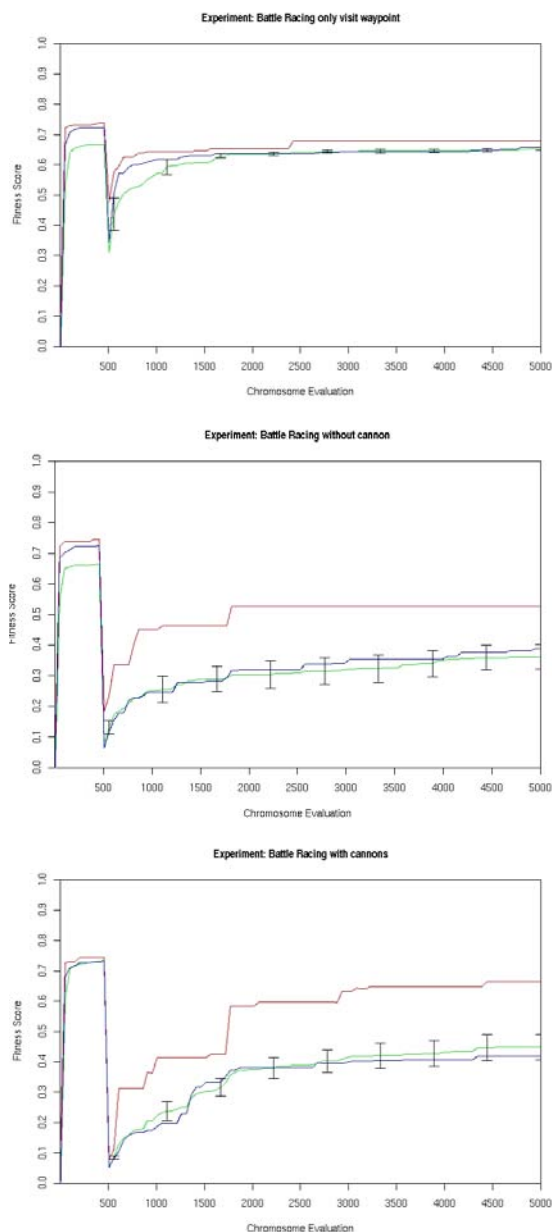


Fig. 7. Graphs presenting the best runs of our waypoint race problem. We see in this instance that utilising the destroy target controller to eliminate enemy players can garner strong results almost as good as those without it. However the difficulty is evident throughout the learning process.

VI. DISCUSSION

The results from this paper have provided some interesting insight and answered the questions we raised in Section III.

- *Would an iterative re-training allow for agents lower in the architecture exploit controllers above it to improve performance?*

Our results in Figure 4 and Table I suggest that this is the case. By giving an extra cycle to improve the controllers once completed, we see improvement that surpasses the results of the traditional approach on

average.

- *While our controllers were relatively small and fast to train, if we broke them apart using subsumption would we see any improvement?*

Results shown in Tables II and III indicate that modifications can result in improvement in the final fitness (albeit marginal) as well as stabilise the learning process. The resulting behaviours also appear more robust and less subject to erroneous decisions.

- *Can we take the goal-centric (base) controllers and use the subsumption to generate agents capable of dealing with conflicting goals?*

From our data gathered in Table IV and Figure 7, there is data to suggest that this in fact possible. Despite not being able to surpass the scores of the *visit waypoint* racer, our aggressive 2-layer controller can generate similar fitness, while providing an entirely different strategy to the problem.

The answers provided are satisfying, suggesting there is potential in further exploration of our subsumption approach. The results shown throughout this paper as well as our analysis of the resulting papers show that there could be potential in larger applications beyond EvoTanks. We fear that the restricted scope of this environment does not give us sufficient breadth to explore whether statistical significance can be achieved. The use of iterative training may provide means to circumvent issues that would arise while using a single training phase for each controller. Further training cycles could allow the learning process to explore new means to overcome issues that were not present when the controller was originally trained, since the added complexity to the environment may have been responsible for the difficulties. An alternative application is to tailor the lower controllers to respect those above it within the hierarchy; since all controllers are tailored to respect those beneath as a result of the training method. In future work we wish to explore specific problem instances and controller combinations where these circumstances may arise, and whether we could utilise this approach to ease any difficulties the agent may have.

Results in increasing modularity seem positive, however we feel the problem instances and controllers we have do not suffer sufficiently to truly explore the potential of this approach. It is important to note that the changes result in either

near identical results or bring around minor improvements. We would be interested in exploring this method against controllers where the provided input is strongly desired by the designer, yet results in unfavourable alterations to the agents behaviour. Whether placing that input in its own subsumption layer would bring about improvement remains to be seen.

Finally, the use of layered de-coupled goal-driven controllers shown in the racing examples is encouraging. Our initial designs in [1] suggested that this type of approach would not prove effective, even the initial work in [8] avoids exploring this type of control. We wish to conduct further research on this form of control. Researchers may also take interest in creating reactive controllers that can switch between two different behaviours depending on the current problem and stimuli fed to the agent. This leaves us with the potential to generate more flexible reactive agents that carry a greater functionality than our previous designs.

VII. CONCLUSION

In this paper we explore the use of a decentralised agent architecture that incorporates neural networks within a subsumption hierarchy. We have raised a series of questions regards the use of iterative training, increased modularity and use of decoupled controllers. The results provide an interesting reflection on the use of this architecture and how it may benefit agent designers, as well as how we could possibly utilise this in future research to greater effect.

REFERENCES

- [1] T. Thompson and J. Levine, "Scaling-up Behaviours in EvoTanks: Applying Subsumption Principles to Artificial Neural Networks," *Computational Intelligence and Games, 2008. CIG 2008. IEEE Symposium on*, 2008.
- [2] R. Brooks, "A robust layered control system for a mobile robot." *IEEE J. ROBOTICS AUTOM.*, vol. 2, no. 1, pp. 14–23, 1986.
- [3] —, "Planning is Just a Way of Avoiding Figuring Out What To Do Next," *MIT Artificial Intelligence Laboratory Working Paper 303*, 1987.
- [4] —, "Elephants Don't Play Chess," *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pp. 3–15, 1991.
- [5] —, "Intelligence without Representation," *Foundations of Artificial Intelligence*, MIT Press, Cambridge, MA, pp. 139–159, 1992.
- [6] T. Thompson, J. Levine, and G. Hayes, "EvoTanks: Co-Evolutionary Development of Game-Playing Agents," *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pp. 328–333, 2007.
- [7] T. Thompson, "EvoTanks ii," Master's thesis, Department of Computer and Information Sciences, Glasgow, Scotland, 2005.
- [8] J. Togelius, "Evolution of a subsumption architecture neurocontroller," *Journal of Intelligent and Fuzzy Systems*, vol. 15, no. 1, pp. 15–20, 2004.