Real-Time Strategy Game Micro for Tactical Training Simulations

Sushil J. Louis, Tianyi Jiang, Siming Liu
Evolutionary Computing Systems Lab
University of Nevada, Reno, Nevada
sushil@cse.unr.edu

ABSTRACT

Complex, realistic scenarios in training simulations can benefit from good control of large numbers of simulation entities. However, training simulations typically focus on simulation physics and graphics over the intelligence required to control large numbers of entities. Real-Time Strategy games, on the other hand, have evolved to make tradeoffs between the AI needed and human interaction required to control hundreds of entities in complex tactical skirmishes. Borrowing from work in real-time strategy games, this paper attacks the problem of controlling groups of heterogenous entities in training simulations by using a genetic algorithm to evolve control algorithm parameters that maximize damage done and minimize damage received during skirmishes in a real-time strategy game-like simulation. Results show the emergence of complex, coordinated behavior among groups of simulated entities. Evolved behavior quality seems to be relatively independent of the underlying physics model but depends on the initial dispositions of entities in the simulation. We can get over this dependence and evolve more robust high performance behaviors by evaluating fitness in several different scenarios with different initial dispositions. We believe these preliminary results indicate the viability of our approach for generating robust, high performance behaviors for controlling swarms of entities in training simulations to enable more complex, realistic training scenarios.

ACM Reference Format:
Sushil J. Louis, Tianyi Jiang, Siming Liu. 2018. Real-Time Strategy Game Micro for Tactical Training Simulations. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3205651.3208288

1 INTRODUCTION

Our research focuses on Real-Time Strategy (RTS) games which map to a broad category of strategic and tactical planning problems in defense applications. The game genre has become a popular research platform for the study of Computational and Artificial Intelligence (CI and AI). In RTS games, players need to establish bases, collect resources, and train military units with the aim of eliminating their opponents. RTS games have a number of properties that make them interesting for AI research [7, 27]. The dynamic environment of RTS games requires strategic, tactical, and reactive planning under uncertainity provided by a fog of war. There is no one optimal strategy; for every strategy, there is a counter, and misleading the opponent is one of the keys to winning. Figure 1 shows a screenshot from Starcraft 2, the most popular RTS game.

Underlying every RTS game is a simulation engine simulating the physics of entities, combat, and game outcomes. If we consider more realistic physics for entities and better battle damage assessment, an RTS game is a wargaming simulation and can be used for simulation training.

In RTS games, the term "Macro" specifies the strategic planning dealing with economy building and choosing the types and numbers of units to produce. "Micro" refers to the fine-grained, nimble, player control of units and unit groups during a skirmish to inflict maximum damage on the enemy while sustaining minimal damage to friendly units. In this paper, we map micro to control of simulated teammates and opponents in training simulations. Althought, most RTS games only allow maneuvering in two dimensions, we attack the problem of generating good micro for realistic 3D maneuvering in simulations. RTS games, being primarily for entertainment,

Figure 1: A screenshot of gameplay in Starcraft 2, a popular RTS game

make tradeoffs between human and AI roles in controlling entities. For example, consider a scenario where we would like to have one group of units harass an opponent group of units. In an RTS game, a player would have to rapidly execute several mouse and keyboard commands in order to harass the opponent. In competition and for

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208288

fun, gaining such fast micro skills helps differentiate players and RTS games therefore tradeoff simpler AI for more human control. Although simulation training can take advantage of interaction design and the simple unit AI in RTS games, training applications usually need to make the opposite tradeoff. We want to reduce the micro burden on the trainer by increasing AI competence. That is, unlike in an RTS game where heavy player micro harrases the opponent, simulation training would prefer providing *harass* as an actual command to be invoked by the trainer.

We use a genetic algorithm to tune the parameters of a *kiting* algorithm for RTS games that corresponds well to a harass algorithm in simulation training for groups composed from two types of entities. Results show that by tuning these parameters we can evolve robust, high performing harassing (and other) behavior. Our approach can evolve such behaviors for different sets of entity properties, works in two and three dimensions, and under a variety of starting initial conditions. These preliminary results indicate the viability of our approach for providing transparent AI for controlling training simulation entities. Videos at https://bit.ly/2IpNFjb show the kinds of complex behavior that emerge.

The next section provides an overview of related work. We describe our control algorithms, representation, and underlying simulation environment in Section 3. Experimental results and discussion follow. The last section provides conclusions and future work.

2 RELATED WORK

Good micro for skirmishes in battle or more generally, coordinated group behavior for adaptive agents, has applications in many fields from wargaming and training to modeling industrial agents, robotics, simulations, and video games [4, 8, 10, 14, 16, 19, 20, 24, 26, 29, 32]. Although flocking, swarming, and other distributed control algorithms [8, 29] drawn from observing natural behavior have been studied, we are interested in simulation training for tactical and strategic battles rather than the precursor navigation and movement control that these algorithms provide. In RTS games, there is strong research interest in generating good micro since this can lead to your winning skirmishes even when your forces are outnumbered or otherwise disadvantaged. This paper focuses on evolutionary computing approaches to generating coordinated autonomous behaviors (micro) for groups of simulation units (or entities). Specifically, we use a genetic algorithm for searching through the space of kiting algorithm parameters to find parameter sets that lead to good micro. The kiting algorithm specifies how to select targets, when to run away, when to run back, and how and where to move as a group. Since the genetic algorithm searches through the space of the kiting algorithm's parameters, we consider it a meta algorithm and name our approach meta search.

Within the broader gaming community, Yannakakis [33] evolved opponent behaviors while Doherty [12] evolved tactical team behavior for teams of agents. Although these approaches are viable, they have only been studied in two dimensions and without the range of unit properties in our work. Many other approaches use potential fields and influence maps to direct movement and so we describe such prior work next.

In physics, a potential field is usually a distance dependent vector field generated by a force. Well designed combinations (usually linear vector sums) of several attractive and repulsive potential fields can lead to fairly complex observed behavior [6]. Potential fields were introduced into robotics as a method for real-time obstacle avoiding navigation and have been used in many video games for controlling multi-unit group movement [14, 21, 26, 29]. In this context, Hagelback and Johnson used potential fields to drive micro in ORTS, a research RTS and partial clone of Starcraft, a very popular commercial RTS [2, 17]. Using some of this earlier work, meta search evolves potential field parameters associated with friendly units as they maneuver during skirmishes.

An influence map structures the world into a 2D or 3D grid and assigns a value to each grid element or cell. Early work in RTS games used influence maps for spatial reasoning to evolve a *LagoonCraft* RTS game player [25]. Much other work uses influence maps for spatial reasoning in games and there are multiple approaches to designing influence maps [5, 9, 28, 30]. Closer to our work, Uriarte used influence maps for generating kiting behavior for their Starcraft player, named *Nova*'s, micro [31]. Our approach when played against Nova, learned to tune algorithm parameters to beat Nova [22]. In our 3D experiments with two types of units called zealots and vultures (borrowed from Starcraft), eight vultures (fragile, fast, longer ranged units) learn to disperse and kite a much larger number (50) of baseline zealots (robust, relatively slow, short ranged units) while suffering relatively little or no damage. Furthermore, we show that our approach is not dependent on the physics used by the units by exploring several different sets of unit speeds, accelerations, and turn rates. Results show that we can evolve high performance micro under several different sets of physics parameters, and that the evolved parameter values adapt reactive behaviors to these physics parameters well. Finally, when groups composed of both zealots and vultures fight against equivalent (same numbers of zealots and vultures) opponents, zealots and vultures learn different strategies and learn to cooperate in defeating opponents.

The next section describes our simulation test-bed, and the 3D physics model for unit movement. Section 4 specifies our representation and describes the parallel genetic algorithm used to evolve control algorithm parameters. We then provide and discuss our results. The last section draws conclusions and points out directions for future work.

3 SIMULATION ENVIRONMENT

Our research platform is an easily parallelizable, speed adjustable, naval simulation built on top of the Ogre3D graphics engine [1, 3]. Figure 2 shows a screenshot from our simulator. Note that our simulation's graphics engine runs on a separate thread and can be turned off, a useful feature for use with evolutionary computing techniques. We modeled game play in FastEcslent to be similar to StarCraft but with a navy-relevant, open ocean setting.

In our experimental scenarios, each player controls a group of units modeled from StarCraft and each group fights against the other within a limited time duration. The units used in our simulation reflect vultures and zealots in StarCraft. A vulture moves fast with a ranged weapon but is vulnerable with low hit-points which makes it easy to destroy. A zealot is a melee unit and moves less fast

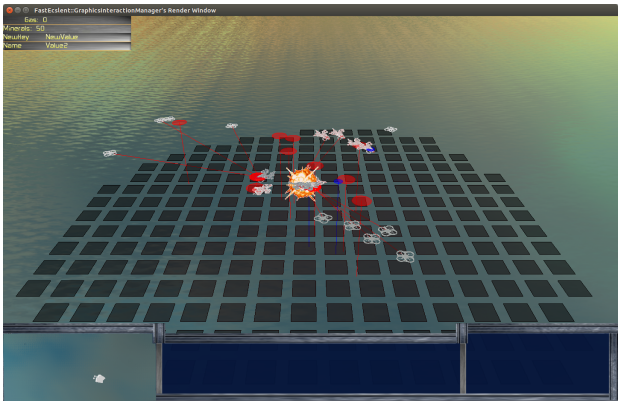

Figure 2: A screenshot from FastEcslent.

Table 1: Unit properties defined in FastEcslent

Property	Vulture	Zealot
Hit-points	80	160
MaxSpeed	64	40
MaxDamage	20	16×2
Weapon's Range	256	224
Weapon's Cooldown	1.1	1.24

but has high hit-points and so is harder to destroy. Table 1 shows properties of interest for both the vulture equivalents and zealot equivalents used in FastEcslent. Since our research focuses on skirmish scenarios suitable for decision making training simulations, we disabled "fog of war" in our experiments.

Game units are controlled by providing them a desired heading (d_h), a desired altitude (d_a), and a desired speed (d_s). We use the following physics equations to control units in our simulation. An entity's current speed depends on the entity's acceleration r_s, simulation time step δt, and desired speed d_s

$$s = s \pm r_s \times \delta t$$

where the $\pm$ depends on whether the desired speed is less than ($-$) or greater than ($+$) current speed. Speeds are clamped to stay between 0 and a maximum speed that depends on the type of entity. We set a maximum altitude for units to 1000 (and minimum of 0) and used a climb rate, r_c, for every unit. Like for speed, the heading and altitude of an entity given by

$$h = h \pm r_t \times \delta t$$

and

$$a = a \pm r_c \times \delta t$$

where h represents heading, a is altitude, r_t is rotation speed, and r_c is climbing rate. From speed, heading, and altitude, we then compute 3D entity velocity ($\vec{v}$) and position ($\vec{p}$) as follows

$$\begin{aligned} \vec{v} &= (s \times cos(h), 0, s \times sin(h)) \\ \vec{p} &= \vec{p} + \vec{v} \times \delta t \\ \vec{p}.y &= a \end{aligned} \tag{1}$$

Here, the xz-plane is the horizontal plane, the y-coordinate is altitude, and we assume the entity is facing toward its heading on the xz-plane.

We represent enemy spatial information using a 3D Influence Map (IM) to provide possible move-to locations. The 3D enemy IM is specified by two parameters, *weight* and the *range*. We use a cell size of $64 \times 64 \times 64$ pixels to reduce the computational load of updating

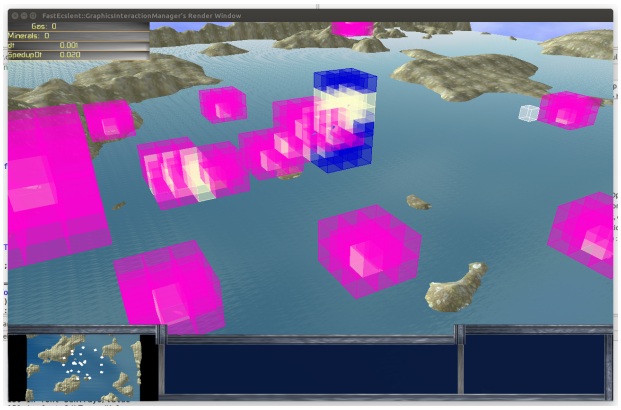

Figure 3: A 3D influence map. Cubes represent unit influence with blue for friendly and red for enemies. The rest of our figures and movies only show the 2D projection of this map on the water surface so we can actually see unit actions

the 3D IM. A unit occupying an influnce map cell gives the cell the value *weight* and this influence extends to *range* neighboring cells in all three dimensions. The value at a cell is then the sum of all units that influence that cell. The GA evolves the weight and the range parameters of the 3D IM and units move towards the cell with lowest value. We update the 3D enemy IM over multiple frames within a total of three seconds to avoid slowing down our simulations. Once we know where to move, we use potential fields to move our group to that location.

Equation 2 describes the standard Potential Field (PF) function.

$$\vec{F} = c\vec{d}^{e} \tag{2}$$

Here $\vec{F}$ is the 3D potential field used by the entity, with $\vec{d}$ being the distance vector from the enemy entity. c and e are evolvable parameters representing the coefficient and the exponent. We use one attractor PF and one repulsor PF to control entity movement.

$$\vec{PF} = c_a\vec{d}^{e_a} + c_r\vec{d}^{e_r}$$

where c_a and e_a are attractor force parameters, and c_r and e_r parameters for the repulsor force. Once our forces get close to the opponent, a parameterized reactive control algorithm taken from [11] controls unit movement and targetting. These reactive control algorithm parameters are then tuned by the genetic algorithm to generate high performance micro that maximizes fitness. Algorithm 1 specifies the control algorithm with the targeting and kiting portions outlined. The algorithm has three major parts. First, R_{nt} specifies a radius around the current target within which to find the next target. Second, a single parameter HP_{ef} controls targeting and tells us when to target a specific enemy unit as opposed to the

Algorithm 1 Reactive Control Algorithm

```
UpdatePosition();
nearbyUnits ← FindNearbyUnits(enemies, R_nt);
highFocusUnit ← GetHighFocusUnit(nearbyUnits);

// Targeting
if lowUnit.healthPercentage < HP_ef then
   Target ← lowUnit
else if GetNumberOfAttackers(highFocusUnit) > 0 then
   Target ← highFocusUnit
else
   Target ← closestUnit
end if

// Kiting
if Weapon.cooldownTimer < (S_t * Weapon.cooldownRate) then
   return
end if
if Weapon.cooldownTimer ≤ 0 then
   IM ← GetEnemyIM();
   Unit ← IM.getLowestValueUnit();
   UpdateSquadState();
   if distanceFrom(Target) > seekRange then
      UnitAI.SetCommand(MoveTowards(Unit));
      return;
   end if
   UnitAI.SetCommand(Attack(Target));
else
   KitingPos ← IM.GetKitingPos(position, Target.position, D_kb)
   if distanceFrom(Target) < (Target.Weapon.range + D_k) then
      if Weapon.range > Target.Weapon.range then
         MoveTowards(KitingPos)
      else if BeingTargetedBy(enemies) and healthPercentage < HP_fb
      then
         MoveAwayFrom(Target);
      end if
   end if
end if
```

most high value unit or the closest unit. Third, four parameters determine kiting behavior. S_t is the freeze time after each firing. D_{kb}, what distance to move away when kiting and D_k, the distance from a target at which units begin kiting. Finally, HP_{fb} determines when units flee.

The genetic algorithm thus evolves the values of a total of 12 parameters. Two 3D IM parameters, four PF parameter, and six reactive control parameters. These parameters are encoded into a 51-bit string chromosome.

The control algorithm is not hard to understand and therefore can be validated by training instructors and subject matter experts. However, finding the values of these parameters for a specific opposing force can be very time consuming and not a good use of time for a human. A genetic algorithm maximing damage done and minimizing damage received enables quick parameter tuning and saves valuable subject matter expert time.

4 REPRESENTATION

We use the following following fitness function to evolve friendly vulture's control parameters against a fixed set of enemy zealots.

$$fitness = damage + (HP \times 400) \quad (3)$$

damage represents the total damage, in hitpoints, dealt by friendly units to enemy units and HP is the total remaining hit-points of all friendly units at the end of a game. This equation seeks to maximize damage done to opponent units while minimizing damage received by friendly units. The 400 weight balances the relative worth of vultures and zealots and enables vultures to avoid over aggressive combat.

When evaluating a member of the genetic algorithm's population, we use the parameter set specified by the individual to control friendly entities in our simulation against an opponent AI. The simulation runs for a fixed time or until one side loses all units. We then compute the fitness function and return the fitness to the genetic algorithm. Since we have to run the simulation for every individual for every generation of the algorithm, we used a simple parallel master slave genetic algorithm in which all individuals are evaluated in parallel. This was an elitist genetic algorithm where offspring compete with each other as well as their parents for populations slots [15, 18]. We used a population size of 50 and run the parallel GA for 60 generations. The probability of 4-point crossover was set to 0.88 and the probability of bit-mutation was 0.01. These values were experimentally determined to work well.

We implemented a baseline AI to control the enemy zealots in all scenarios used in this study. Enemy zealots controlled by the baseline AI will move toward and attack the closest opponent unit. In these experiments, eight (8) vultures skirmished against 50 baseline zealots. Initial results on experimental scenarios where each player controls a group of units initially positioned in opposite corners on a map without any obstacles or neutral entities were encouraging. With the two groups of units on opposite sides of the map and the baseline AI controlling zealots, meta search evolved effective micro for 3D vultures achieving on average 93.6% of maximum possible fitness over 30 runs of the genetic algorithm with different random seeds [23].

5 RESULTS

We did two kinds of experiments to show the robustness of our meta search approach and suitability for simulation training applications. In the first set of experiments we investigated the effect of changing unit physics parameters. Good performance under different maneuvering characteristics not only indicates that the approach works across different physics, but also shows which physics parameters have the largest effect on effectiveness. All code for all our expirements is at https://bit.ly/2IpNFjb along with videos showing the emerging entity behaviors. Table 2 shows the the two physics

Table 2: Physics Parameters

Property	$P1$	$P2$	$P3$	Description
S_m	64	64	64	Maximum Speed
R_m	60	75	90	Maximum Rotation Speed
A_m	55	85	115	Maximum Acceleration

parameters that we considered varying. R_m, the rotation speed and A_m, the acceleration. Varying the maximum speed significantly affects vulture effectiveness and we left it at a fixed value.

We get the theoretical maximum performance when our vultures suffer no damage and eliminate all zealots. This works out to a theoretical maximum fitness value of 19200. Figure 4 shows the overall

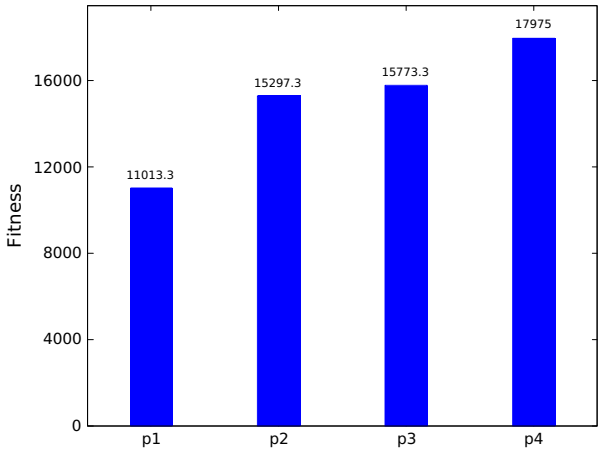

Figure 4: Average of maximum fitness of the AI player using the realistic physics $P1$, $P2$, $P3$, and the game-like physics $P4$ over 30 runs.

performance of the evolved players using physics $P1$, $P2$, and $P3$. We also considered unrealistic game physics with instantaneous turns and the ability to reach maximum speed instantaneously. The bar labeled $P4$ denotes performance under this kind of physics. The results show that $P4$ units without realistic physical limits have, as might be expected, the highest performance (93%). When comparing units moving under more realistic physics, as in training simulations, we find that high values of rotation speed and acceleration $P3$ and $P2$ performed similarly to $P4$ with only a 10% relative performance loss. Both eliminate almost all the opponent zealots while losing only one vulture at the end of the skirmish. Finally, slower accelerating and turning units ($P1$) do worst. Over several other experiments, we also find that acceleration is more important than turning rate in our simulation because units need not face the target to fire. All these results are averages over 30 runs of the genetic algorithm.

These results show that the meta search approach works across multiple physics implementations and is able to tune an understandable (transparent) control algorithm to these different physics characteristics. However, our approach has two drawbacks. First, we require an existing opponent, the equivalent of our baseline AI, to evolve against. We are working on co-evolutionary meta-search to tackle this issue [13]. Second, we have not fully addressed the question of robustness. That is, does this approach work with groups composed of multiple unit types and across multiple different scenarios?

We attack these issues by investigating groups composed from vultures and zealots skirmishing against an identical opponent group in five different scenarios. The GA in this case tunes a total of 24 parameters, 12 for vultures and 12 for zealots. We encode all parameters into a $51 \times 2 = 102$-bit string and run the simulation to return a fitness. In this study, Equation 4 determines fitness.

$$fitness = Damage_z + Damage_v + 160H_{fz} + 400H_{fv} \quad (4)$$

$Damage_z$ and $Damage_v$ are the total damage done by friendly zealots and vultures, H_{fz} and H_{fv} are the remaining hit point (HP) of friendly zealots and vultures. Again, the equation aims to maximize damage done to opponent units and minimize damage received by friendly units. The vulture in StarCraft requires more resources than the zealot. Hence, we weight vulture as 400 and zealot as 160 based on prior research [11].

As pointed out earlier, we get within 93% of optimal on average over 30 runs when evolving vultures against a baseline zealot AI (ZAI). However, in this work, each side has two types of units: zealots and vultures and we need a baseline AI that controls both types of units to evolve against. We thus evolved the zealot and vulture opponent AIs separately. To generate a good opponent zealot AI, we used the approach in [23] to evolve a good zealot AI by evolving zealot micro parameters against ZAI. The best individual over 30 runs against ZAI, became our opponent zealot AI.

We then evolved a vulture opponent AI against this new opponent zealot AI. As before, the best individual over 30 runs against the opponent zealot AI, became our new opponent vulture AI. To test the performance of the new opponent AI, we ran it against 3000 random enemies and obtained an average fitness of 55% of maximum. Anything over 50% usually means that we won the skirmish.

The five scenarios are described below. Our side is blue (dark) and the opponent is white[1] and we start with the Line Formation scenario shown in Figure 5. This deployed

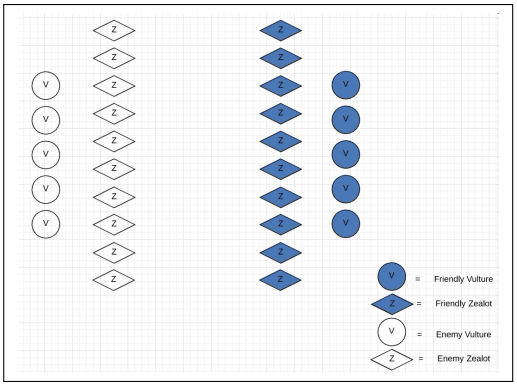

Figure 5: Line Formation

units in a line formation with 10 zealots in front and 5 vultures behind. Such a formation puts durable units (zealots) in front of fragile units (vultures) to protect fragile units, and enable valuable fragile units to do more damage and stand longer. Second, the Surround scenario surrounds blue force with white as shown in Figure 6.

[1]We wanted red versus blue but settled for white versus blue since both red and blue are dark colors and difficult to differentiate in black and white print.

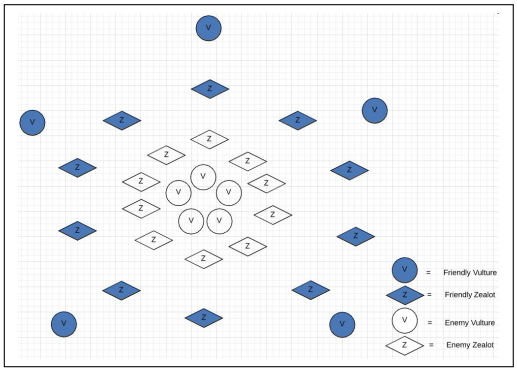

Figure 6: Surround

Third, the Trapped scenario surrounds white force with blue and is the inverse of Surround. In the fourth scenario, called Random, all units started with random position and orientation.

When evolving in each of these scenarios, fitness is evaluated only on the scenario. Our fifth and last scenario is different and we evaluate fitness by evaluating chromosomes in each of the four scenarios above and summing the fitnesses. All skirmishes pit a group composed of 5 vultures and 10 zealots against an identical opponent.

In our first experiment with heterogeneous groups, we start by setting the blue group on the right side and the enemy group on the left side. We then ran the GA 30 times with 50 population size and for 60 generations. Figure 7 shows that our GA can find an average of maximum fitness over 30 runs of 5510 out of a possible maximum fitness of 7200 which is 5510/7200 = 76.5% of maximum. Note again that values above 50% mean that blue beat white.

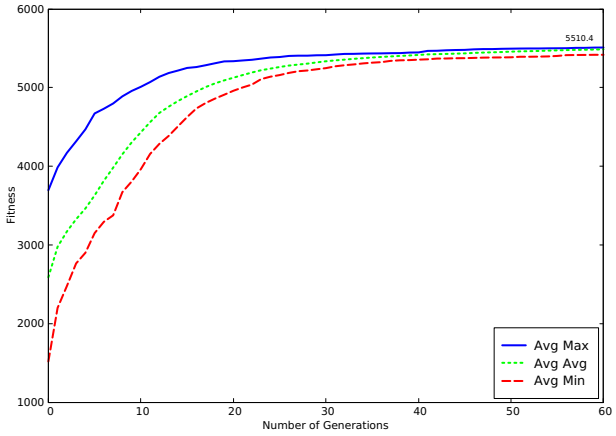

Figure 7: GA performance on Line Formation.

Blue surrounds white in the second scenario as shown in Figure 6. This scenario specifies an aggressive stance with a high probability of destroying all enemies in a short time period. The GA settings are the same as in the previous experiment. Figure 8 shows our GA can find an average maximum fitness of 5182 which is 5182/7200 = 72% of maximum.

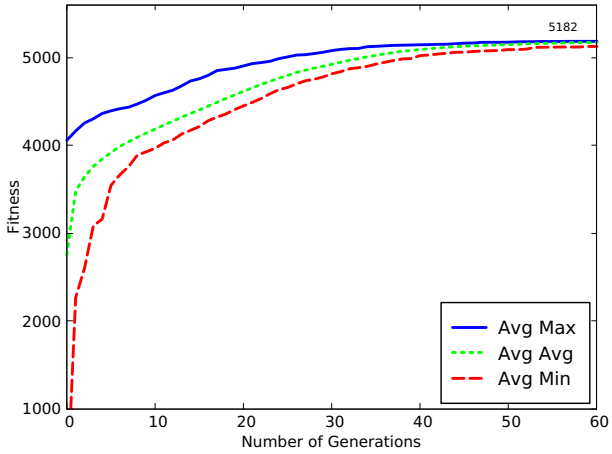

Figure 8: GA performance on Surround.

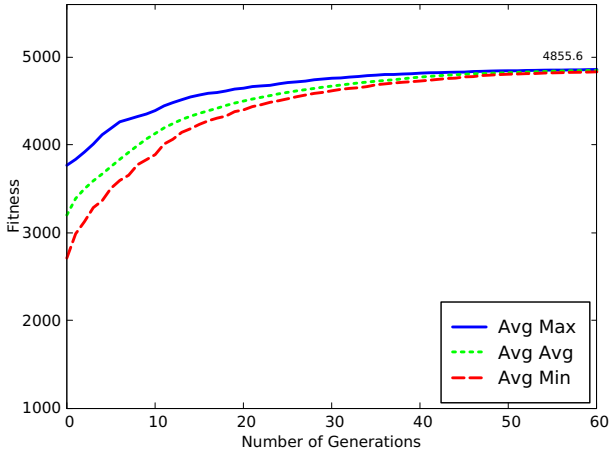

Figure 9: GA on combined scenario.

The GA converges smoothly in the third experiment as well and we get to 74% of maximum when white surrounds blue. Despite random positioning being rare in simulations and in games, we generate randoms positions for all units initially and then use the same set of randomly generated initial positions in each run of the GA. The GA does relatively poorly and barely manages to win against white reaching 51.5% of optimal on these random scenarios.

The results from these first four experimental scenarios show that we are able to achieve performance between 50% and 80% of maximum. Note that although we know that 7200 is the theoretical maximum fitness achievable, when evolving against the opponent AI, we do not expect our units to survive unscathed and get to the theoretical maximum. In order to inflict damage, our units have to take damage and any fitness over 50% of maximum implies that we destroyed all enemy units while some of our units survived.

We can now test how a solution evolved in one scenario performs on another scenario and evaluate solution robustness. We chose the best individual from all 30 runs on one scenario and tested this

Table 3: Performance on training and testing scenarios.

	Line	Trapped	Surround	Random	Avg on others
Line	79%	48%	52%	59%	53%
Trapped	37%	80%	57%	52%	48.7%
Surround	42%	63%	77%	55%	53.3%
Random	45%	71%	68%	77%	61.3%
Combined	77%	63%	67%	71%	69.5%

Table 4: Average performance across 50 different random scenarios.

Scenario	Avg over 50 random scenarios
Line	47%
Trapped	46%
Surround	43%
Random	49%
Combined	56%

individual's robustness in other scenarios. The first four rows of Table 3 show these results. If we consider the scenario in which a solution evolves as the training scenario and the rest as testing scenarios, the boxed values show performance on training scenarios. We can see that good performance on the training scenario does not guarantee good performance on test scenarios. In addition, none of the individuals do well on the Random scenario. These results indicate that generalization remains elusive and we address this issue next.

We did a fifth set of experiments where we combined all four scenarios into one fitness function so that we can find a solution that generalizes better. Figure 9 shows that the GA can find a maximum average fitness of 4855 which is 4855/7200 = 67% of maximum. The last row of Table 3 shows that the best solution from the combined scenario does the best across all scenarios including random. These results suggest that we need to *train* in diverse scenarios to evolve robust solutions that perform well in never-before seen scenarios.

To confirm this, we tested each of the best solutions from the five experiments on 50 different scenarios where all units are randomly positioned. Again we can see that the solution from the combined scenario does statistically significantly better. This testing provided even stronger evidence that evolving in a variety of scenarios results in more robust performance.

6 CONCLUSIONS AND FUTURE WORK

This paper shows that genetic algorithms can reliably evolve high quality, robust, 3D micro behaviors for entities under multiple physics models for single and multi unit type groups. The genetic algorithm tunes the parameters of our transparent control algorithm that uses potential fields to guide unit movement and uses influence maps to determine vulnerable positions to attack. The representation and evolutionary approach seems to generalize well and produces different high performing micro behaviors tuned to the physics of the entities being controlled.

The paper also shows that the approach extends to multi-unit groups and we find that training on multiple scenarios leads to more robust solutions that perform better on new test scenarios. Compared to solutions that only trained on one scenario, solutions trained on multiple scenarios performed significantly better on 50 randomly generated scenarios. Videos at https://bit.ly/2IpNFjb show some of these solutions and the complexity of the behavior that emerges.

We currently require an opponent AI to evolve against. To eliminate this requirement, we plan to investigate a co-evolutionary approach to generate robust control for groups of units in simulations and in games. In addition, we are interested in investigating applying meta-search to groups of unmanned vehicles and to groups composed from many types of units.

7 ACKNOWLEDGEMENTS

This work was in part funded by grant N00014-15-1-2015 and N00014-17-1-2558 from the Office of Naval Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.

REFERENCES

[1] 2002. Ogre3d. (2002). https://www.ogre3d.org/
[2] 2010. Starcraft. (2010). http://us.battle.net/sc2
[3] 2016. Fast Evolutionary Computing Systems Lab ENTity engine. (2016). https://ecsl.cse.unr.edu
[4] M. Barbuceanu and M.S. Fox. 1995. COOL: A language for describing coordination in multi agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*. Citeseer, 17–24.
[5] Maurice Bergsma and Pieter Spronck. 2008. Adaptive spatial reasoning for turn-based strategy games. *Proceedings of AIIDE* (2008).
[6] Valentino Braitenberg. 1986. *Vehicles: Experiments in synthetic psychology*. MIT press.
[7] Michael Buro. 2003. Real-time strategy games: A new AI research challenge. *Proceedings of the 18th International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence* (2003), 1534–1535.
[8] Y.L. Chuang, Y.R. Huang, M.R. D'Orsogna, and A.L. Bertozzi. 2007. Multi-vehicle flocking: scalability of cooperative control algorithms using pairwise potentials. In *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2292–2299.
[9] Holger Danielsiek, R Stuer, Andreas Thom, Nicola Beume, Boris Naujoks, and Mike Preuss. 2008. Intelligent moving of groups in real-time strategy games. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 71–78.
[10] P. Dasgupta. 2008. A multiagent swarming system for distributed automatic target recognition using unmanned aerial vehicles. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 38, 3 (2008), 549–563.
[11] Tyler DeWitt, Sushil J Louis, and Siming Liu. 2016. Evolving micro for 3d real-time strategy games. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*. IEEE, 1–8.
[12] D. Doherty and C. OâĂŹRiordan. 2006. Evolving tactical behaviours for teams of agents in single player action games. In *Proceedings of the 9th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games*. 121–126.
[13] R. Dubey, J. Ghantous, S. Louis, and S. Liu. 2018. Evolutionary Multi-objective Optimization of Real-Time Strategy Micro. *ArXiv e-prints* (March 2018). arXiv:1803.10316
[14] M. Egerstedt and X. Hu. 2001. Formation constrained multi-agent control. *Robotics and Automation, IEEE Transactions on* 17, 6 (2001), 947–951.
[15] L.J. Eshelman. 1991. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. *Foundations of genetic algorithms* (1991), 265–283.
[16] J. Ferber and O. Gutknecht. 1998. A meta-model for the analysis and design of organizations in multi-agent systems. In *Multi Agent Systems, 1998. Proceedings. International Conference on*. IEEE, 128–135.
[17] J. Hagelbäck and S.J. Johansson. 2008. Using multi-agent potential fields in real-time strategy games. In *Proceedings of the 7th international joint conference on*

Autonomous agents and multiagent systems-Volume 2. International Foundation for Autonomous Agents and Multiagent Systems, 631–638.

[18] John Henry Holland. 1992. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[19] N.R. Jennings. 1993. Commitments and conventions: The foundation of coordination in multi-agent systems. *Knowledge Engineering Review* 8 (1993), 223–223.

[20] N.R. Jennings. 1995. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial intelligence* 75, 2 (1995), 195–240.

[21] O. Khatib. [n. d.]. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research* 5 ([n. d.]).

[22] S. Liu, S. Louis, and C. Ballinger. 2016. Evolving Effective Micro Behaviors in Real-Time Strategy Games. *IEEE Transactions on Computational Intelligence and AI in Games* PP, 99 (2016), 1–1. https://doi.org/10.1109/TCIAIG.2016.2544844

[23] Siming Liu, Sushil J Louis, Tianyi Jiang, and Rui Wu. 2017. Increasing physics realism when evolving micro behaviors for 3D RTS games. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2465–2472.

[24] M.J. Matarić. 1995. Designing and understanding adaptive group behavior. *Adaptive Behavior* 4, 1 (1995), 51–80.

[25] C. Miles, J. Quiroz, R. Leigh, and S.J. Louis. 2007. Co-Evolving Influence Map Tree Based Strategy Game Players. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*. 88 –95. https://doi.org/10.1109/CIG.2007.368083

[26] R. Olfati-Saber, J.A. Fax, and R.M. Murray. 2007. Consensus and cooperation in networked multi-agent systems. *Proc. IEEE* 95, 1 (2007), 215–233.

[27] Santiago Ontañon, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, Mike Preuss, et al. 2013. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. *IEEE Transactions on Computational Intelligence and AI in games* 5, 4 (2013), 1–19.

[28] Mike Preuss, Nicola Beume, Holger Danielsiek, Tobias Hein, Boris Naujoks, Nico Piatkowski, Raphael StuĬLer, Andreas Thom, and Simon Wessing. 2010. Towards intelligent team composition and maneuvering in real-time strategy games. *Computational Intelligence and AI in Games, IEEE Transactions on* 2, 2 (2010), 82–98.

[29] C.W. Reynolds. 1987. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, Vol. 21. ACM, 25–34.

[30] Penelope Sweetser and Janet Wiles. 2005. Combining influence maps and cellular automata for reactive game agents. *Intelligent Data Engineering and Automated Learning-IDEAL 2005* (2005), 209–215.

[31] Alberto Uriarte and Santiago Ontañón. 2012. Kiting in RTS Games Using Influence Maps. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[32] P. Vadakkepat, K.C. Tan, and W. Ming-Liang. 2000. Evolutionary artificial potential fields and their application in real time robot path planning. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, Vol. 1. IEEE, 256–263.

[33] G.N. Yannakakis and J. Hallam. 2004. Evolving opponents for interesting interactive computer games. *From Animals to Animats* 8 (2004), 499–508.