Evolving Micro for 3D Real-Time Strategy Games

Tyler DeWitt, Sushil J. Louis, and Siming Liu
Dept. of Computer Science and Engineering
University of Nevada, Reno
{tdewitt, sushil, simingl}@cse.unr.edu

***Abstract*—This paper extends prior work in generating two dimensional micro for Real-Time Strategy games to three dimensions. We extend our influence map and potential fields representation to three dimensions and compare two hill-climbers with a genetic algorithm on the problem of generating high performance influence map, potential field, and reactive control parameters that control the behavior of units in an open source Real-Time Strategy game. Results indicate that genetic algorithms evolve better behaviors for ranged units that inflict damage on enemies while kiting to avoid damage. Additionally, genetic algorithms evolve better behaviors for melee units that concentrate firepower on selective enemies to decrease the opposing army's effectiveness. Evolved behaviors, particularly for ranged units, generalize well to new scenarios. Our work thus provides evidence for the viability of an influence map and potential fields based representation for reactive control algorithms in games, 3D simulations, and aerial vehicle swarms.**

I. Introduction

Real-Time Strategy (RTS) games are a sub-genre of video games where players gather resources to build units to fight and defeat adversaries. Players collect resources to power up an economy that can then produce military units used to destroy opponent units and economy. RTS games thus incorporate elements of strategic economic development and tactical battle management that adds complexity to game play. As such, many interesting CI and AI research challenges exist within the game genre [1], [2], [3]. First, dynamic environments within RTS games mean that we need real-time planning on several levels - strategic, tactical, and reactive. Second, a "fog of war" hides enemy disposition and strategy, therefore players have to scout to gain information to formulate effective strategies. Third, players must learn and exploit their opponents' playing "style" quickly in order to gain the advantage in future games. Fourth, players must employ spatial and temporal reasoning to coordinate effective unit formations and time-sensitive actions on a tactical and strategic level.

These challenges lead to two broad areas of RTS AI research that encapsulate game play elements in RTS games, macro and micro. Macro refers more to long term decision making dealing with resource management and creating a strong economy. A stronger economy enables more production of military units that battle the opponent. Micro refers to controlling small sets of such military units in combat. Good micro minimizes damage received by friendly units while maximizing damage dealt to enemy units. Good macro combined with good micro wins RTS games. Often, an RTS game may have multiple skirmishes between opposing groups of units and superior micro during a single skirmish may change the entire course of the game.

Influence maps (IMs) and potential fields (PFs) techniques have been used, in the past, for spatial reasoning and unit maneuvering [4]. IMs map a number to each cell of a discretized game map and the number can indicate areas of interests within the level to the game AI [5], [6]. In this paper, we extend influence maps to 3D as a volumetric grid over 3D space. Each grid volume, or cell, contains a numerical value indicating the influence of nearby entities. Figure 1 shows the three-dimensional (3D) influence map of enemy units. Each IM cell value, computed by an IM function, depends on two parameters, a `weight`, corresponding to the influence of the entity occupying the cell and a `range` for this influence. The final value at a cell is the sum of the influences of all entities that have that cell within their range. Assume we design the IM function to provide low values for enemy influence and high values for friendly influence, a very low value in certain cells reveals that there is heavy enemy presence and therefore the area corresponding to those cells is dangerous for our units.

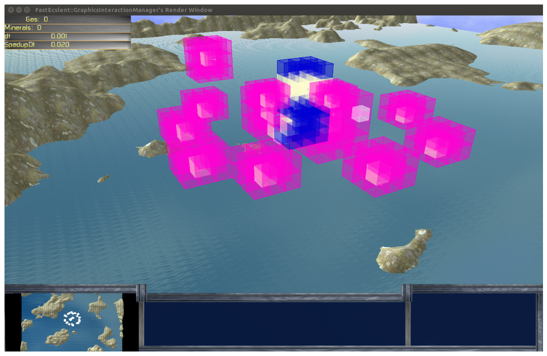


Fig. 1: A 3D influence map showing color coded influence over a map. Pink values are higher than blue which are higher than white. The white areas with the lowest values are thus areas to be avoided by opponents.

Potential field approaches from robotics have been extensively used in guiding group movement in RTS and other games [7], [8], [9]. They enable real-time, cohesive movement, with collision avoidance and have been used to generate good positioning for attack and defense. We extend prior work with 2D potential fields to three dimensions and evolve the (now) 3D parameters that define attractive and repulsive potential fields for game units.

In our experiments, we use a 3D influence map generated from enemy units to tell our units where to go and use two 3D potential fields to control unit navigation. Earlier work has shown that influence maps and potential fields provide representations that can be used by parameterized, but simple, reactive control algorithms to generate high performance 2D micro [9]. In this work, good kiting, targeting, and fleeing behaviors evolved from the tuning of 3D IM, 3D PF, and 3D reactive algorithm parameters.

This paper extends Liu's 2D work to 3D [9]. Specifically, we use and compare genetic algorithms (GAs) and two hill climbers (HCs) on the problem of finding good 3D influence map, 3D potential field, and reactive control parameters that lead to high performance 3D micro. We generate and compare micro performance in our simulation environment with units similar to Zealots, a close-in, melee unit, and Vultures, a fast, ranged unit, in StarCraft. Our Zealots and Vultures have the ability to move in three dimensions, that is, units can fly in 3D space. We also report on micro performance on scenarios not used during search to investigate how our approach generalizes. Finally, we use FastEcslent, an open source game engine that supports full 3D unit movement in games [10]. We chose FastEcslent in place of the popular StarCraft: Brood Wars API (BWAPI) [11] in order to change the physics and enable full 3D movement. Not only do we want to move to 3D game-physics, but we would like to investigate the effect of more realistic physics on evolved "micro" performance for real-world unmanned aerial vehicles. Figure 2 shows a screen shot of in-game combat between two teams of units within FastEcslent.

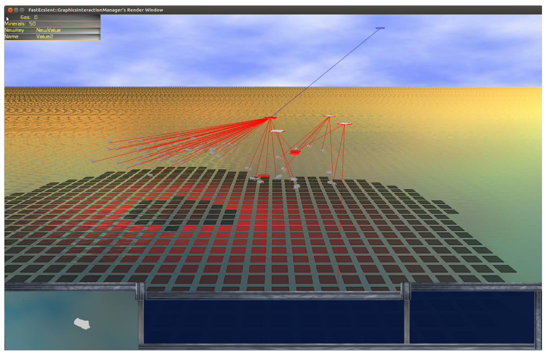


Fig. 2: Units are able to fly and fight enemies in 3D within FastEcslent. Note that although the influence map appears 2D, only the bottom layer of influence map cells is being rendered to provide an unobstructed view of the skirmish.

Preliminary results indicate that both GAs and HCs are able to evolve competent 3D micro. Ranged units (Vultures) learn to kite against melee units (Zealots) in three dimensions and spread firepower by moving to split the enemy unit group into smaller subgroups and thus avoid being surrounded. Ranged units evolve conservative behaviors that preserve health and do not engage in risky tactics. Melee units, when trained against evolved ranged units, learn to concentrate firepower on individual ranged units and diminish the damage dealt by the enemy unit group. Results also show that GAs more consistently produce higher quality solutions than HCs. That is, although hill-climbers occasionally generate high performance micro in shorter time, they do so unreliably. On the other hand, GAs may take more time to produce near-optimal solutions but do so more reliably.

The remainder of this paper is structured as follows. Section II discusses related work in RTS AI research, as well as common micro implementations and approaches. Section III explains our simulation environment, the design of our AI player, and detailed IM and PF representation for generated 3D micro. Section IV shows preliminary results and compares the solutions generated by the search algorithms and investigates the applicability of solutions in new scenarios. The final section draws conclusions and explores future work.

II. Related Work

Numerous techniques have been used in the design of RTS AI players[3]. However, we focus on work related to micro management including spatial reasoning and unit movement. In this context, influence maps have been a popular technique for spatial reasoning in RTS and other games. Sweeter *et al.* designed a game agent that used IMs and cellular automata to model the game environment and assist the agent in decision-making within their EmerGEnT game world [5]. Their game agent was capable of pursuing a target while responding to both user actions and natural phenomenon. Bergsma and Spronck implemented IMs to produce adaptive AI for combat in turn-based strategy games [6]. Their adaptive AI evolved high level decision making using an evolutionary algorithm. Avery *et al.* used IMs to co-evolve tactics for a group of boats to move autonomously and attack enemy units in co-ordination [12]. Their technique generated an IM for each unit in order to produce different unit navigation; however, their technique was computationally intensive when increasing the number of units in-game. Preuss *et al.* generated group movement by using flocking in combination with IM path finding within the RTS game *Glest* [13], [14]. Their approach found improvements in group performances across each of their game scenarios. For our research we evolve and use an enemy IM to collect spatial information on enemy disposition and to direct friendly units to good locations from which to launch attacks. Potential fields then guide unit movement during attacks.

PFs were first introduced by Ossama Khatib as a computationally simple approach to real-time obstacle avoidance for mobile robots [7]. This method was then extensively used for collision avoidance among multiple entities [15], [16], [17]. In games, most work related to PFs involve spatial navigation and collision avoidance [18]. Multi-agent potential fields were used by Hagelbäck and Johansson for unit navigation and obstacle avoidance in RTS games [8]. Their research involved the development of an AI player that incorporated PFs at the tactical and reactive control level [19]. Early work in our lab applied spatial reasoning techniques with IMs to evolve a complete RTS game player [20]. More recent work combined IMs and PFs as a basis representation to generate micro position and movement tactics [9], [21], [4]. In this paper, we extend Liu's reactive control algorithm for micro to use 3D IMs and PFs.

Previous work by Uriarte and Ontañón implemented kiting using IMs for group positioning [22]. Their approach was incorporated into the NOVA bot, which competed in the annual StarCraft AI Competition. Gunnerud *et al.* developed a hybrid system that combines case-base reasoning and reinforcement learning which improves itself while playing an RTS game. The hybrid system learned effective targeting behaviors specific for a given scenario [23]. Wender *et al.* examined the suitability of reinforced learning algorithms for executing effective micro in the RTS game StarCraft: Brood Wars [24]. Their results indicate that reinforcement learning algorithms are capable of developing strategies for winning small scale battle while learning actions such as “Fight”, “Retreat”, and “Idle” in combat. Limitations exist in their implementation however, as the default StarCraftBW AI was the opponent and performance was evaluated on a limited set of tasks. In this work, we extend and use Liu’s parameterized stateless distributed control algorithm which tries to maximize damage to the enemy while minimizing the amount of damage received by friendly units. With appropriate evolved parameters, the algorithm generated 2D kiting, targeting, and fleeing. We extend the algorithm to work in 3D to investigate the evolution of similar behaviors. Although many RTS games do not support full 3D movement, our work does, and as such may also be applicable to user interaction with, and control of, real 3D unmanned aerial vehicle swarms.

III. METHODOLOGY

We used FastEcslent, an open-source and research-oriented game engine built on OGRE [25] that supports full 3D entity movement. Since graphics and user interaction run within a separate thread, FastEcslent can run without graphics or interaction, enabling easier integration with heuristic search algorithms. On the other hand, even when copying unit parameters from StarcraftBW, trying to replicate StarCraft movement and combat exactly is non-trivial. However, exact duplication is not necessary to evaluate our approach and to investigate whether we can evolve kiting and other well know micro behaviors during combat. With that caveat, our skirmish scenarios built within FastEcslent reflect combat found in StarCraftBW by replicating StarCraft units and their respective properties. In addition, we implement 3D physics and extend our influence map and potential field implementations to 3D. Figure 2 shows an in-game screen shot of a FastEcslent skirmish scenario between two opposing sides. In our scenario, each player controls a group of units initially positioned in opposite corners. In our experimental scenarios, the map does not contain obstacles or neutral entities. Second, FastEcslent entity properties reflect those of default StarCraft units, more specifically, Vultures and Zealots. A Vulture is a ranged unit with low hit-points but high movement speed, proving to be effective when outmaneuvering slower enemy units. A Zealot is a melee unit (low attack range) that has more hit-points than a Vulture but is comparatively slower. Table I details the parameters for both Vultures and Zealots in FastEcslent. Lastly, there is no fog of war since we are only looking at skirmishes, not a complete game. We also implemented a baseline opponent AI that behaves similar to the default StarCraft AI to control enemy Zealots. The maximum running time for our scenario is 6000 frames, approximately one minute at normal game speed. We created a skirmish scenario with four 3D moving Vultures on side RED (our side) and fifty 3D moving Zealots on side BLUE.

A. Influence Maps and Potential Fields

We represent group behavior as a combination of one enemy influence map, attractor and repulsor potential fields, and a set of reactive control parameters. The IM provides possible move-to locations and the PFs control movement to locations provided by the IM. Two parameters, the `weight` and `range` specify the IM. Since computation time also depends on the number of IM cells in the map, we use a cell size of $64 \times 64 \times 64$ pixels in the game map. If an enemy unit occupies a cell, the value of that cell and all neighboring cells in `range` get `weight` added to their current value. We call this the SumIM and `weight` and `range` are the evolvable parameters. Since we are evolving micro in a full 3D environment in this paper, influence maps and potential fields extend to three dimensions as well. However, extending IMs from 2D to 3D increases the computational complexity of their implementation. The original 2D IM was of $O(MN)$ complexity where M is the number of IM cells on the x-axis and N is the number of IM cells on the y-axis. Considering the number of cells for the original 2D implementation of FastEcslent (64×64), 4096 cells updates were needed to update the IM. Since entities now move in three dimensions, the introduction of the z-axis increases the computational complexity to $O(MNL)$, where L is the number of IM cells on the z-axis. However, our IM implementation updates IM cells over multiple frames within a total of three seconds and does not noticeably slow down simulations or adversely affect unit behavior.

TABLE I: Unit parameters defined in FastEcslent

Parameter	Vulture	Zealot
Hit-points	80	160
Size	45×10×12	18×3×6
MaxSpeed	64	40
MaxDamage	20	16
Weapon’s Range	256	224
Weapon’s Cooldown	1.1	1.24

Equation 1 shows a standard potential field function, where F describes the potential force applied to the entity, with D being the 3D distance from the enemy entity. The force direction is in the direction of the vector difference from the enemy entity and c and e are evolvable parameters.

$$F = cD^e \tag{1}$$

We use one attractor PF and one repulsor PF of the form described by Equation 1 to control entity movement in-game. The attractor force guides a unit towards its target. The repulsor force repels units from other units or obstacles. Normally it is stronger than the attractor force at short distances while being weaker at long distances.

$$\vec{PF} = c_a D^{e_a} + c_r D^{e_r} \tag{2}$$

where c_a and e_a are attractor force parameters, and c_r and e_r parameters for the replusor force.

B. Reactive Controls

Along with group positioning and unit navigation, we represented our reactive control behaviors in a way that our search algorithms can tune. Our reactive control behaviors included micro behaviors frequently used by professional human

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 $\leq 0$ then
    MoveTowardsAndAttack(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
            MoveTowards(Target);
        end if
    end if
end if
```

players: kiting, targeting, and fleeing. Algorithm 1 specifies the algorithm with the targeting and kiting portions outlined.

Targeting selects and concentrates fire on a specific unit depending on candidate enemy unit hit-points and distance. Each of our $units$ selects the nearest enemy unit $t_{closest}$ as a possible target. Within a distance R_{nt} from $t_{closest}$, our $unit$ will select a target based on prioritized criteria. The highest priority is t_{lowhp}, the enemy with the lowest hit-points below the evolvable threshold: HP_{ef}. Next in priority is t_{focus}, the enemy unit being targeted by the most friendly units within R_{nt} relative to $t_{closest}$. The third prioritized criteria is $t_{closest}$, the nearest enemy unit. Kiting serves as a useful hit-run-repeat tactic for units with higher speed and attack range. Units strike quickly and retreat back to avoid being attacked by the slower units. During kiting, our $unit$ moves towards and attacks its $target$ as soon as the $unit$'s weapon is ready, which is dependent on S_t. A $unit$ will begin kiting if the $unit$'s weapon is not ready and if the distance between the $unit$ and its $target$ is less than D_k. The $unit$ moves back (away from the target) to a $kitingPosition$ which is computed by the function $getKitingPositionFromIM(D_{kb})$ from the SumIM. D_{kb} represents the number of cells away from the $target$'s cell. If $Cell_t$ is the IM cell containing our target, $getKitingPositionFromIM(D_{kb})$ finds a neighboring IM cell with the lowest value. We set this new IM cell to $Cell_t$ and then repeat this process of finding a new $Cell_t$, D_{kb} times. The algorithm then uses $Cell_t$ as the $kitingPosition$ to move towards. Finally, fleeing to avoid further damage gets triggered when the unit's hit-points fall below below a threshold, represented by HP_{fb}. HP_{fb} controls this "fleeing" behavior.

We encode 12 micro parameters, consisting of 6 reactive control parameters as well as 6 IM and PF parameters, into a 51-bit binary string that represent a chromosome for our search algorithms. Our search algorithms then decode these encoded binary strings into a set of parameters as shown in [21]. FastEcslent receives this decoded set of parameters and uses them to run the skirmish. When finished, FastEcslent returns the resulting score and fitness to the calling search algorithm.

C. Fitness Evaluation

The goal of our scenario is to maximize enemy unit damage while minimizing friendly unit damage. The evaluation function to compute fitness F reflects this:

$$F \quad = \quad TD_{eu} + (HP_{fu} \times 400) \tag{3}$$

where fitness is calculated at the end of the scenario. TD_{eu} represents the total damage given to enemy units, while HP_{fu} is the sum of remaining hit-points of all friendly units. According to our prior experiments, we use the scalar value 400 for multiplying HP_{fu} to give unit hit-points more weight than enemy unit damage as a means to encourage health conservation and more evasive kiting behaviors. This is somewhat arbitrary and an alternative approach in our current research, is to use a multi-objective evolutionary algorithms and treat damage done and damage received as two criteria in a pareto-optimization setting. Note also that the same fitness can be found in multiple ways. For example, a fitness of 7200 can describe an outcome of 45 enemy units destroyed with no friendly units remaining or an outcome of 40 enemy units destroyed and two friendly units alive with full hit-points. The fitness is used by our search algorithms to bias search.

D. Hill-climbers

The Bit-Setting Optimization (BSO) hill climber searches a locally optimal solution in the search space by sequentially flipping each bit and saving the better fitness solution when it is found [26]. BSO searches a subset of the search space based on the initial point set by the random seed. In order to make results obtained from GAs and HCs comparable, the maximum number of evaluations made by all algorithms are set to the same number, 600. The BSO starts from the left again when it reaches the end of the chromosome. The Random Flip Optimization (RFO), an alternative hill-climber, randomly chooses a bit in the randomly generated initial chromosome and flips it. This is repeated 600 times.

E. Genetic Algorithm

We use an elitist GA in our experiments. Assuming the population size is N, during selection our elitist GA selects the N best individuals from the combined parent and offspring populations ($2N$) to create the next generation after recombination. We implemented a parallel version of this elitist GA where evaluations are done in parallel to significantly speed up our runs. For our scenarios, the population size was 20, run for 30 generations for a total of 600 evaluations. In order to relate our experiment to that of the original 2D implementation, we use the same crossover and mutation rate for our GA.

The probability of our two-point crossover is 88% and bit-flip mutation probability is 0.01. Standard roulette wheel selection is used to select chromosomes for crossover. These operator choices and GA parameter values were empirically determined to work well.

IV. Results and Discussion

Unit AI behavior within FastEcslent is deterministic, meaning that a given set of parameters guarantees the same fitness every time a scenario runs. The FastEcslent game engine does not add any noise to picking a targeting, probability of hitting the target, or in the amount of damage done. We ran our scenarios with 30 random seeds for each search algorithm.

A. Search Algorithm Results

In the skirmish scenario with 4 friendly Vultures vs. 50 enemy Zealots, all three search algorithms were able to evolve high fitness 3D micro within 600 evaluations. Ranged units evolved kiting behaviors that were successful in destroying numerous enemy units while avoiding damage. According to the fitness function, the theoretical maximum score for our scenario is 9600. This is obtained when eliminating all of the enemy units (8000) with no friendly units receiving damage (1600). Figure 3 illustrates the average fitness from 30 runs.

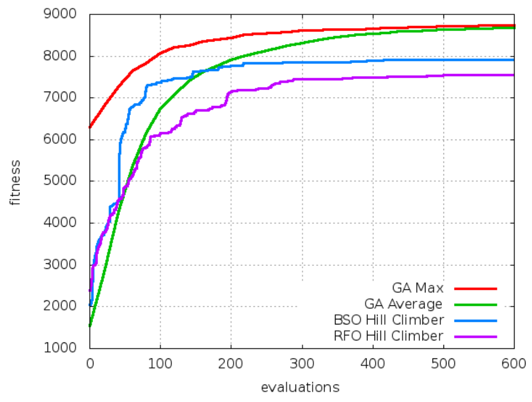


Fig. 3: Average performance of GA, BSO, and RFO for our scenario with 30 different random seeds. X-axis represents number of evaluations and Y-axis shows the average fitness over 30 runs.

The average fitness of the BSO shown in Figure 3 climbed quickly in the first 100 evaluations which seemed to indicate that the BSO quickly finds (local) optima. The final average score for BSO, 7904 was the second highest average score among the three tested algorithms. The best fitness over 30 runs was 9200, indicating that the BSO sometimes did very well and this solution destroyed 48 of the 50 enemy Zealots but did receive some damage. The average fitness of the RFO shown in Figure 3 did not climb as quickly as the other search algorithms and RFO usually did worse than the others. This was reflected in the final average score of 7556 attained over the 30 RFO runs, the worst among the three tested algorithms. On our scenario, RFO seemed less reliable than BSO. However, the highest fitness obtained by RFO was 9020 which indicated that RFO also has the potential to find relatively high quality solutions. The solution with a score of 9020 destroyed 43 units but received more damage than the best solution found by the BSO.

The average fitness curves of the GA shown in Figure 3 rise smoothly and end higher than the averages of both HCs. The average over 30 runs of the maximum fitness in the GA population (GA Max) was also consistently higher than the quick climbing BSO. This indicated that the GA was more reliably and more quickly find high fitness solutions. The final average fitness for the GA was 8745.3 which was the highest average among the three tested algorithms. The highest fitness obtained by the GA was 9260 which was also the highest fitness found by any algorithm. This solution inflicted 7660 damage, destroying 42 of the 50 Zealots while not receiving any damage. The solution produced by the GA destroyed the least amount of Zealots out of all search algorithms, but displayed near-optimal kiting abilities by avoiding any damage whatsoever.

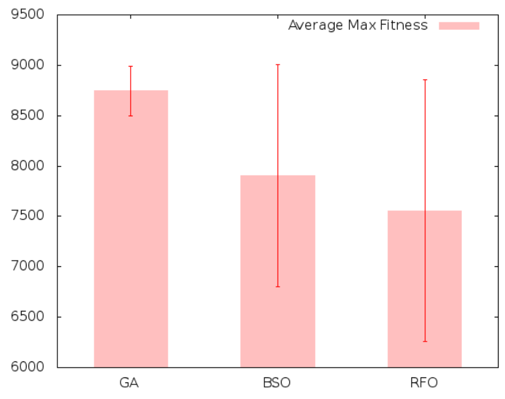


Fig. 4: For our scenario, the standard deviation showed by the error bars tells that GA on average produced more reliable solutions after 600 evaluations.

If we define a fitness of 8000 as our threshold for good performance, we can see that the GA performed better than either HC. The GA found solutions above 8000 every run. The BSO could only find solutions above a score of 8000 on 16 out of the 30 runs while the RFO found solutions above a score of 8000 on 13 of the 30 runs. The differences in final average fitness between the BSO and GA were statistically significant with a one-tailed $P < 0.0001$. Additionally, Figure 4 shows that the standard deviation of the GA's set of final fitnesses was 245.17, whereas the standard deviation of the BSO's final fitnesses was 1105.43. The differences in final average fitnesses between the GA and RFO were statistically significant with a one-tailed $P < 0.0001$. The standard deviation for RFO final solutions was 1298.26. These statistically significant results provide evidence that the GA more reliably produces higher quality 3D micro.

B. Evolved 3D Micro Behavior

We are also interested in the highest fitness 3D micro behavior generated by the search algorithms. The parameters for evolved Vultures in Table II produced by the GA results in a score of 9260, the highest score found. The behavior created by these parameters enabled friendly Vultures to spread across the map and split enemy units into smaller subgroups, thus decreasing the concentrated firepower of the more numerous enemy group so our units do not become overwhelmed. Figure 5 shows a screen shot of the skirmish that illustrates our Vulture's 3D micro behavior. The parameters specifying PF values show that our units were strongly attracted towards enemy units with small repulsion, allowing friendly units to strike closely but remain out of the enemy unit's weapon range. A low freeze time (S_t) also allows for units to kite more frequently and avoid becoming overwhelmed when at stand still. A maximum value of the HP_{ef} parameter demonstrates that the evolved Vultures did not prefer targeting previously damaged enemies and instead targeted units closest to them, preventing potentially dangerous chases through enemy groups and kiting in quick, successive intervals. Videos of evolved micro compared with initially generated micro can be found online at http://www.cse.unr.edu/~tdewitt/.

TABLE II: Best found solutions for both units.

Unit	IMs		PFs				Reactive control					
	W	R	c_a	c_r	e_a	e_r	St	D_k	R_{nt}	D_{kb}	HP_{ef}	HP_{fb}
Vultures	14	10	60	13	10	3	2	22	14	4	7	1
Zealots	12	9	55	27	9	2	6	21	10	6	5	7

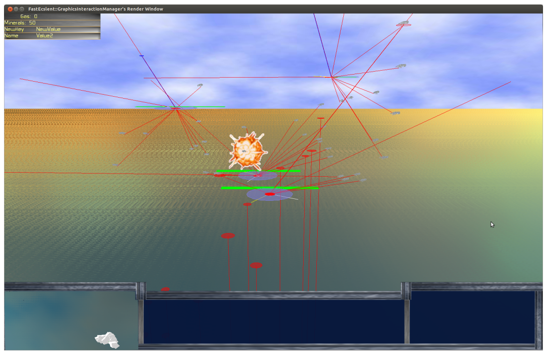


Fig. 5: Vultures with evolved micro fight smaller fragments of the enemy group for a higher chance of survival.

C. Generalizability of Evolved 3D Micro Behaviors

We tested the generalizability of our evolved set of parameters for Vultures found in Table II by applying them to control vultures in new scenarios. For each side, at its corner, we randomly generated the unit positions of each side and averaged the fitnesses over 500 runs. Figure 6 (red bars) shows the fitness distribution of all 500 runs for this generalizability test. The average score out of 500 runs is 6391 which is 69.02% of the highest fitness evolved in the original scenario (9260).

We then further tested the generalizability of the evolved Vulture micro by randomly generating initial unit positions anywhere within the map and averaged the scores from each run. Figure 6 (blue bars) also shows the distribution of fitnesses over all 500 runs on this scenario. The average fitness is 6588 which is 71.14% of the highest fitness evolved in the original scenario. Although parameters were evolved on one specific scenario with fixed initial positions for all units, our representation leads to behavior that is somewhat generalizable over other initial positions.

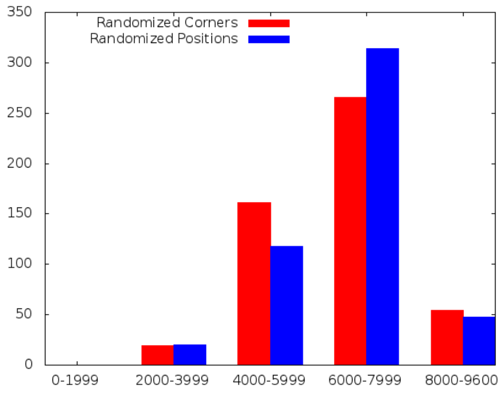


Fig. 6: The evolved 3D micro from our scenario of fitness 9260 generalizes well to new scenarios.

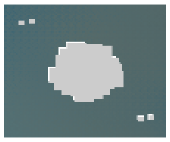
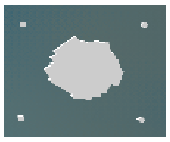
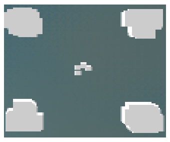
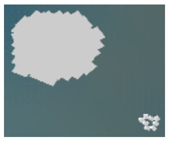
We also apply our solution to new scenarios with fixed initial positions. Table III details the design and results of the new scenarios used to test the kiting efficiency of our evolved Vultures. Scenario 1 starts our 4 evolved Vultures separated into two subgroups in opposite corners to surround 50 enemy Zealots placed in the center of the map. Vultures immediately split the Zealots into two groups with each group moving towards the closest Vulture subgroup. We lost 1 Vulture early in the scenario which decreased the overall fire power of our units for the remaining time duration, resulting in only 19 enemy units destroyed for a fitness of 6160. Scenario 2 places each of the 4 Vultures in each corner with 50 Zealots placed in the map's center. Zealots immediately split into four groups and began to move towards the Vulture closest to them. Vultures were already separated across the map and quickly engaged in kiting behaviors, eliminating 35 of the Zealots with no casualties. This scenario results in a fitness of 8060. Scenario 3 inverts the previous scenario by placing the Zealots in the corners and Vultures in the map's center. Our evolved Vultures split to individually fight Zealot subgroups at the beginning of the scenario. We lose 1 Vulture halfway through the scenario but it was alive long enough to eliminate multiple enemy units, therefore the loss in overall fire power was not as severe as if it had been eliminated early on. Our evolved Vultures still managed to eliminate 31 enemy units, resulting in a fitness of 7220. Scenario 4 doubles unit numbers and has 8 Vultures versus 100 Zealots. Vultures were able to handle Zealots well and although we lost 1 Vulture towards the end of the scenario, the Vultures destroyed 65 Zealots.

The evolved Vultures still engage in some risky behavior and casualties from this, result in a decrease in overall fire power for the entire Vulture group and lowers the group's tactical effectiveness by a considerable amount. However, Vultures still perform well by spreading across the map and kiting Zealot subgroups effectively.

D. Evolving 3D Zealot Micro Behavior

Once we had good Vulture micro, we investigated evolving 3D Zealot micro against these previously evolved Vultures.

TABLE III: Screen shot of initial 3D unit positioning for four new scenarios.

Scenario	Description	Results
	4 Vultures in opposite corners versus 50 Zealots in center.	Fitness of 6160. 1 friendly destroyed, 19 enemies destroyed.
	4 Vultures in each corner versus 50 Zealots in center.	Fitness of 8060. 0 friendlies destroyed, 35 enemies destroyed.
	4 Vultures in center versus 50 Zealots divided into each corner.	Fitness of 7220. 1 friendly destroyed, 31 enemies destroyed.
	8 Vultures versus 100 Zealots.	Fitness of 14080 1 friendly destroyed, 65 enemies destroyed.

To do so we replicated the same set of experiments with the same map rules while swapping unit sides, therefore our new scenario now consists of 50 friendly Zealots fighting 4 enemy Vultures controlled by the highest performing micro in Table II found in our original scenario. We also modify our fitness function to better suite the objective of melee attack units in our new scenario. The new evaluation function is:

$$F = \begin{cases} (D_{eu} \times 100) + (N_{fu} \times 100), & \text{if } N_{eu} = 0 \\ (D_{eu} \times 100) + (N_{fu} \times 10), & \text{otherwise} \end{cases}$$

where D_{eu} represents damage, the sum of enemy unit casualties. N_{fu} and N_{eu} are the number of friendly units and number of enemy units remaining at the end of the scenario. With this fitness function there is only one way to achieve a particular fitness. For example, a fitness of 4300 indicates that all enemy units were destroyed with 39 friendly units remaining. This conditional fitness function is to guide search algorithms in evolving micro that eliminates all enemy units first in order to avoid evolving passive micro. Again, in future work, we plan to use multi-objective evolutionary algorithms that try to maximize damage done and minimize damage received as the two criteria to be pareto-optimized.

For this scenario, the theoretical maximum score for the scenario is 5400. This is obtained by eliminating all of the enemy units (400) and retaining all friendly units (5000). Figure 7 shows that we are able to evolve Zealots to fight and win against evolved Vultures in this specific scenario. The GA found solutions that were able to eliminate all enemy units in 22 of 30 runs for an average max fitness of 3451. The highest fitness obtained is 5000, which destroyed all 4 enemy Vulture units while losing 4 of 50 friendly Zealot units.

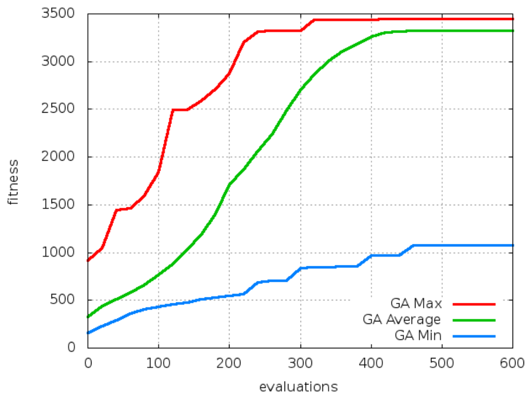


Fig. 7: Average performance of GA for our scenario with 30 different random seeds. X-axis represents number of evaluations and Y-axis shows the average fitness at that evaluation.

The GA was able to generate high quality solutions for melee versus ranged units by having Zealots concentrate firepower as a group on one enemy Vulture at a time, reducing the overall effectiveness of the enemy group with each unit eliminated. Instead of kiting, Zealots learn to rush one Vulture at a time as a means to overwhelm and quickly eliminate this Vulture. Zealots do not evolve kiting behaviors due to their inability to outrun enemy Vultures and instead develop more aggressive, risky behavior to destroy ranged enemy units. Rather than spacing out to fight smaller subgroups of enemy units, Zealots collectively form a condensed group and focus firepower on one enemy unit at a time. The fewer number of Vultures alive, the higher the number of Zealots that stay alive as the skirmish continues. This provides an incentive for Zealots to eliminate Vultures quickly to avoid prolonged skirmishes that lead to more Zealot casualties later in the scenario. A low repulsive PF evolves and allows Zealots to move into a more compact group which permits rushing with the concentrated firepower needed to eliminate ranged enemy units. Table II lists the evolved parameter values for the best evolved Zealot.

Further experiments in testing the generalizability of evolved 3D Zealot micro concludes that there exists limitations in our representation for evolving micro of this specific unit type. Kiting behaviors do not apply well to these melee units when fighting ranged units and our representation of micro does not incorporate effective melee micro parameters (i.e. flanking).

V. Conclusion and Future Work

This paper extends prior work in generating two dimensional micro for Real-Time Strategy games to three dimensions. We use influence maps and potential fields to coordinate group positioning and unit movement during skirmishes. Unit group behavior is represented as a set of parameters that define an influence map, an attractive and a repulsive potential field, and reactive controls while limiting the search space for our search algorithms to 2^{51}. Results show that the genetic algorithm and two hillclimbers can find parameter values that lead to high fitness correlated with good micro. However, the genetic algorithm more reliably and more quickly finds higher fitness parameter values. Both hill climbers find good solutions between 40% and 60% of the time, while the genetic algorithm finds high quality solution a 100% of the time. These results are statistically significant. For ranged versus melee unit combat, ranged units see higher effectiveness when the group becomes more spread and splits enemy firepower. Moreover, ranged units kite efficiently by attacking enemy units while avoiding being within enemy weapon's range. Conversely, evolved melee units can successfully eliminate ranged units by concentrating fire on one unit at a time, quickly reducing the overall effectiveness of the enemy group with each unit destroyed. Our evolved Vultures exploit opposing melee units (Zealots) in every scenario by slicing enemy units into smaller groups to avoid becoming overwhelmed and then kiting till skirmish-time runs out. Results also show that although our evolved 3D ranged unit (Vulture) micro generalize well to new scenarios, our evolved 3D Zealot micro does not generalize as well to other scenarios.

We are interested in evolving effective 3D micro for melee units against ranged units with appropriate representations of melee micro behaviors. We plan to investigate simplifying the fitness functions by turning to a multi-objective formulation of the problem and using multi-objective evolutionary algorithms. Techniques such as case-injection or other knowledge-based systems may be added to our system in future research to further investigate speed, quality, and generalizability of our representation and evolved solutions. We are also interested in co-evolving micro for rather than evolving micro against a fixed opponent. Essentially, we manually did one co-evolutionary cycle when evolving Zealot micro against our prior evolved Vultures. Finally, we plan to investigate evolving multi-unit micro with more complex unit interactions.

Acknowledgment

This material is based in part upon work supported by the National Science Foundation under Grant Number IIA-1301726. 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 National Science Foundation. This work is also partially supported by the Office of Naval Research under Grant N00014-15-1-2015.

References

[1] M. Buro, "Call for ai research in rts games," in *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, 2004, pp. 139–142.

[2] M. Buro and T. Furtak, "Rts games and real-time ai research," in *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, vol. 6370, 2004.

[3] S. Ontanón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game ai research and competition in starcraft," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 5, no. 4, pp. 293–311, 2013.

[4] S. Liu, S. J. Louis, and M. Nicolescu, "Using cigar for finding effective group behaviors in rts game," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[5] P. Sweetser and J. Wiles, "Combining influence maps and cellular automata for reactive game agents," *Intelligent Data Engineering and Automated Learning-IDEAL 2005*, pp. 209–215, 2005.

[6] M. Bergsma and P. Spronck, "Adaptive spatial reasoning for turn-based strategy games," *Proceedings of AIIDE*, 2008.

[7] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.

[8] J. Hagelbäck and S. J. Johansson, "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*, ser. AAMAS '08. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 631–638. [Online]. Available: http://dl.acm.org/citation.cfm?id=1402298.1402312

[9] S. Liu, S. J. Louis, and C. Ballinger, "Using ga for finding effective micro behaviors in rts game."

[10] (2016) Evolutionary computing systems lab, unr. [Online]. Available: http://ecsl.cse.unr.edu/

[11] M. Buro, "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*, pp. 1534–1535, 2003.

[12] P. Avery and S. Louis, "Coevolving influence maps for spatial team tactics in a RTS game," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 783–790. [Online]. Available: http://doi.acm.org/10.1145/1830483.1830621

[13] M. Preuss, N. Beume, H. Danielsiek, T. Hein, B. Naujoks, N. Piatkowski, R. Stuer, A. Thom, and S. Wessing, "Towards intelligent team composition and maneuvering in real-time strategy games," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 2, pp. 82–98, 2010.

[14] H. Danielsiek, R. Stuer, A. Thom, N. Beume, B. Naujoks, and M. Preuss, "Intelligent moving of groups in real-time strategy games," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 71–78.

[15] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[16] M. Egerstedt and X. Hu, "Formation constrained multi-agent control," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 6, pp. 947–951, 2001.

[17] C. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4. ACM, 1987, pp. 25–34.

[18] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 3, pp. 278–288, 1991.

[19] J. Hagelbäck and S. J. Johansson, "The rise of potential fields in real time strategy bots," *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2008.

[20] C. Miles, J. Quiroz, R. Leigh, and S. Louis, "Co-evolving influence map tree based strategy game players," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, april 2007, pp. 88 –95.

[21] S. Liu, S. J. Louis, and M. Nicolescu, "Comparing heuristic search methods for finding effective group behaviors in rts game," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013, pp. 1371–1378.

[22] A. Uriarte and S. Ontanón, "Kiting in rts games using influence maps," in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[23] M. J. Gunnerud, "A cbr/rl system for learning micromanagement in real-time strategy games," 2009.

[24] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: broodwar," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 402–408.

[25] T. K. S. Ltd, "Ogre open source 3d graphics engine," February 2005. [Online]. Available: http://www.ogre3d.org/

[26] S. W. Wilson, "Ga-easy doe not imply steepest-ascent optimizable," 1991.