

## Kiting in RTS Games Using Influence Maps

Alberto Uriarte and Santiago Ontañón

Computer Science Department  
Drexel University  
{albertouri,santi}@cs.drexel.edu

### Abstract

Influence Maps have been successfully used in controlling the navigation of multiple units. In this paper, we apply the idea to the problem of simulating a kiting behavior (also known as “attack and flee”) in the context of real-time strategy (RTS) games. We present our approach and evaluate it in the popular RTS game StarCraft, where we analyze the benefits that our approach brings to a StarCraft playing bot.

### Introduction

Real-time Strategy (RTS) is a game genre where players need to build an economy (gathering resources and building a base) and a military power (training units and researching technologies) in order to defeat their opponents (destroying their army and base). RTS games are real-time, non deterministic, partially observable and have huge state spaces. Therefore, compared to traditional board games, RTS games pose significant challenges for artificial intelligence (Buro 2003). One of such challenges is unit and group control. How to effectively control squads of units or how to simulate complex tactical behaviors in a real time environment is still an open problem in game AI research. Specifically, in this paper, we present an approach based on influence maps to simulate *kiting behavior* (also known as “attack and flee”). This advanced tactical move is specially helpful to handle combats where we are weaker than our enemy but our attack range is bigger than the enemy. In those cases using a kiting behavior is the difference between losing or winning.

There has been a significant amount of work on recreating realistic squad movements. From using the flocking behaviors described by Reynolds (1999) or improving pathfinding using Influence Maps (Tozour 2001), to combining both techniques (Preuss et al. 2010) for RTS games. Those approaches focus on building a robust navigation system, which is still an open problem for real-time games. To solve this problem several solutions have been proposed in several areas. For example, in the area of robotics, Khatib introduced the concept of Artificial Potential Fields (Khatib 1985) to avoid obstacles in real-time for a robot control. Potential fields and Influence Maps have also been found useful for navigation purposes in the domains of robot soccer

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

(Johansson and Saffiotti 2002), or in computer games such as Quake II (Thurau, Bauckhage, and Sagerer 2000) or RTS games (Hagelbäck and Johansson 2008; Avery, Louis, and Avery 2009). However, all of these approaches give more importance to navigation rather than to tactical offensive or defensive moves. Only in their work on potential fields, Hagelbäck and Johansson showed a simple version of the “attack and flee” behavior, but it was not studied in depth.

The work presented in this paper represents a contribution towards achieving more complex and effective tactical moves in real-time domains. Specifically, we propose the use of influence maps (a sister technique to potential fields) to achieve kiting. The approach presented in this paper has been evaluated in the context of StarCraft, and incorporated into the NOVA StarCraft bot (Uriarte 2011) for testing purposes. One of the main disadvantages of potential fields and influence maps is the need to perform parameter tuning. One approach to deal with parameter tuning is to use automatic techniques such as reinforcement learning to converge to good settings (Liu and Li 2008). In this paper we will provide analytic ways to define such parameters whenever possible.

The rest of this paper is organized as follows. First we define the problem statement and some background concepts. Then, we briefly describe StarCraft (our domain) and NOVA (the bot into which we have incorporated our approach). After that, we describe our approach to simulate kiting behavior. Then we present an empirical evaluation of the performance of our approach. Finally, the paper closes with conclusions and directions of future research.

### Problem Statement

This section describes the specific problem that we tackle in this paper. Let us start by introducing the basic concepts of RTS games and *kiting*.

In real-time strategy games, players control an army composed of individual units (tanks, ships, soldiers, etc.). Each of those units can move independently and attack enemy units. Each of such units typically characterized by a collection of attributes such as hit points, movement speed, attack range, and so on. The collection of these attributes that are relevant to the approach presented in this paper will be detailed below, but for now it suffices to give the intuitive notion that a unit can attack another unit only when the target

is inside of its *attack range*; when a unit successfully attacks another, the target’s hit points get reduced in an amount depending on the attack power of the attacker and the armor of the target; and finally, a unit is destroyed when its hit points get below zero.

In the rest of this paper we will distinguish between *friendly units* and *enemy* or *target units* to distinguish the units belonging to the player we are controlling from the units of the enemy player.

The problem we tackle in this paper is how can a friendly unit or group of friendly units attack and destroy an enemy unit or group of enemy units while minimizing the losses amongst the friendly units. Moreover, we will specifically focus on addressing that problem through the simulation of *kiting behavior*.

Intuitively, the basic idea of kiting is to approach the target unit to attack, and then immediately flee out of the attacking range of the target unit. More specifically, given two units  $u_1$  and  $u_2$ , unit  $u_1$  exhibits a *kiting* behavior when it keeps a safe distance from  $u_2$  to reduce the damage taken from attacks of  $u_2$  while the target  $u_2$  keeps pursuing  $u_1$ .

Furthermore, we say that a unit  $u_1$  performs *perfect kiting* when it is able to inflict damage to  $u_2$  without suffering any damage in return, and we say that  $u_1$  performs *sustained kiting* when it is not able to cause enough damage to kill unit  $u_2$ , but  $u_2$  is also unable to kill  $u_1$ .

Sustained kiting is useful in many situations. For example, in many RTS games players need to send units to “scout” what the opponent is doing; these scout units can exploit sustained kiting behavior while scouting the enemy base to remain alive as much as possible, and to make the enemy waste as much time as possible trying to destroy the scout. However, the approach presented in this paper aims at perfect kiting. The execution of perfect kiting depends on the characteristics and abilities of the unit and the target unit. In this paper we will present a method to detect when perfect kiting is possible, and provide an algorithm to execute this behavior successfully.

## StarCraft and NOVA

In this paper we use the game *StarCraft: Brood War* as the testbed for our research. StarCraft is a military science fiction RTS game that is gaining popularity as a testbed for RTS research (Weber 2010). In particular, players in StarCraft can choose between 3 different races (Terran, Protoss and Zerg). Each race has its own units and abilities, requiring different strategies and tactics.

To test our approach we extended the NOVA StarCraft bot with the capability of performing kiting. NOVA uses a multi-agent system architecture (shown in Figure 1) with two types of agents: regular *agents* and *managers*. The difference between regular agents and managers is that NOVA can create an arbitrary number of regular agents of each type, but only one manager of each different type. The different agents and managers shown in Figure 1 have the following function:

- The *Information Manager* is the environment perception of the bot, it retrieves the game state from the game and

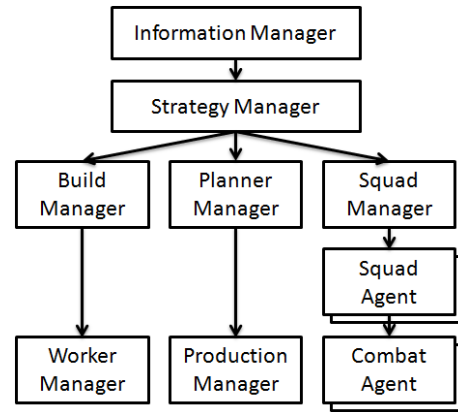


Figure 1: NOVA multi-agent architecture

stores all the relevant information in a shared blackboard.

- The *Strategy Manager*, based on the data from the Information Manager, determines the high level strategy and sends orders to other managers (Build Manager, Planner Manager, and Squad Manager) in order to execute it.
- The *Build Manager* is in charge of creating the different buildings that constitute a base.
- The *Worker Manager* is in charge of controlling all the workers and balance their tasks such as harvesting the different types of resources, or performing scouting tasks.
- The *Planner Manager* is in charge of producing the necessary military units, deciding which are the next units to be produced.
- The *Production Manager* balances the production of new units and research amongst the different buildings capable of performing such tasks.
- The *Squad Manager* is in charge of commanding all the military units to attack the opponent. In order to do so, the Squad Manager spawns one Squad Agent per each group of military units.
- The *Squad Agent* controls the group behavior of one group of military units, assigning them a common target and ensuring they move together as a group.
- The *Combat Agent* controls the low level actions of one military unit, including using the special abilities.

In particular, the approach presented in this paper was implemented as part of the Combat Agent.

## An Influence Map Approach to Kiting

Our approach to model kiting behavior divides the problem in three subproblems: 1) deciding when kiting can be performed, 2) creating an influence map to guide the movement, and 3) target selection. The following three subsections deal with each of those subproblems in turn, and then we present the high-level algorithm that puts them together into the NOVA bot.

Unit	HP	speed	deceleration	acceleration	turn	attackTime	attackRange
Zealot	160	4.0	0	2	1	2	15
Vulture	80	6.4	0	9	3	1	160

Table 1: Vulture and Zealots attributes (times are measured in game frames, and distances in pixels)

### When Can Kiting Be Performed?

It is not always possible to successfully perform a kiting behavior. Some conditions must be met in order to execute it. This section focuses on identifying such conditions. Let us start by introducing some preliminary concepts.

As mentioned above, each unit in a RTS game is defined by a series of attributes, like movement speed, attack range, etc. In particular, in this paper we are concerned with the following attributes:

**speed:** top speed of the unit.

**acceleration:** time that a unit needs to get to top speed.

**deceleration:** time that a unit needs to stop.

**turn:** time that a unit needs to turn 180 degrees.

**attackTime:** time that the unit needs to execute an attack (notice this is different from the “attackCoolDown” attribute that some RTS games have, which defines the minimum time between attacks).

**attackRange:** maximum distance at which a unit can attack.

**HP:** hit points of the unit. When the HP are equal or lower than 0 the unit is removed from the game.

**DPS:** Damage per second that the unit can inflict to a target.

Given two units  $u_1$  and  $u_2$  defined by the previous attributes,  $u_1$  can kite  $u_2$  only when two conditions are met: 1) when  $u_1$  is faster than  $u_2$ , and 2) when  $u_1$  has an attack range long enough so that when it attacks  $u_2$ , it has time to attack, turn around and escape before it is in range of the attack of  $u_2$ . The exact conditions under which those conditions hold might be complex to assess, since they depend on many low-level details of the unit dynamics. However, in general, we can define two simple conditions that, if satisfied ensure that we can perform a successful kiting behavior:

$$u_1.speed > u_2.speed \quad (1)$$

$$u_1.attackRange > u_2.attackRange + u_2.speed \times kitingTime(u_1) \quad (2)$$

where  $kitingTime$  represents the time that  $u_1$  requires to decelerate, turn around, attack, turn around again and leave:

$$kitingTime(u) = u.deceleration + u.attackTime + u.acceleration + 2 \times u.turn$$

Condition 1 ensures that  $u_1$  can increase its distance from  $u_2$  if needed. Condition 2 ensures that  $u_1$  can attack  $u_2$ , turn around and escape, before being inside of the attack range of  $u_2$ . In other words,  $u_1$  can attack and retreat before the enemy unit  $u_2$  can attack. Figure 2 illustrates each of the different actions that consume time during a kiting movement. Notice that in some complex RTS games, such as StarCraft,

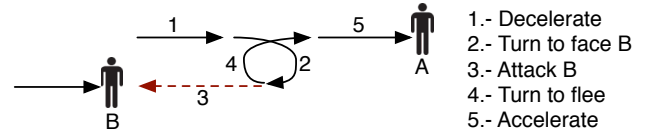


Figure 2: The five actions required when a unit A kites another unit B.

the attributes of the different units are not constant, but can be affected dynamically by some of the unit’s special abilities during the game. For example, Marines (a basic unit in the game StarCraft) can use an ability called “stim packs” in order to increase their *speed* and decrease their *attackTime* for a brief period of time. Thus, the previous equations constitute just a simplification that might need to be adapted for the specific game at hand.

For example, in the case of StarCraft, the previous equations are satisfied when  $u_1$  is of the type *Vulture* (a fast and versatile ranged Terran unit) and  $u_2$  is of the type *Zealot* (a strong close-range ground Protoss unit). Table 1 shows the attributes of both Vultures and Zealot units.

### Influence Maps for Kiting

An *influence map* is a technique to store relevant information about different entities of interest in a game, represented as a numerical *influence*. In the context of RTS games, an influence map can be seen as a two dimensional matrix containing one cell per each position in the game map. An example can be seen on Figure 3, where the influence map contains numerical influence from a unit and from some walls in the map. Influence maps are closely related to the idea of potential fields (Hagelbäck and Johansson 2008), sharing the same underlying principles.

We can use the abstraction of an influence map to keep track of the dangerous areas (where friendly units will be in range of fire of enemy units) or the safe ones. The main idea in our approach is to use influence maps to detect which areas are safe, then, when performing kiting, the friendly unit will first attack the target unit, and then flee to a safe position using the influence map. Once a safe position has been reached, the kiting behavior will be repeated.

An important decision to be made while designing influence maps is the spatial partition. An influence map with a high resolution can be computationally non-viable on real-time environments, but a low resolution one may provide too coarse grained information, making it useless for taking decisions. In the experiments reported in this paper, we used StarCraft as our testbed, which keeps three different game maps at different resolutions: a pixel level map for keeping unit location, a walk tile map (with a resolution of  $8 \times 8$  pix-


	1	1	1	1	1	1	1		
	1	1	4	4	4	4	4		
			3	3	3	4	4		
			3	3		4	4		
			3	3	3	4	4		
			3	3	3	4	4		
						1	1		

Figure 3: Example of Influence Map. The numbers are the threat in that position, a higher value means more danger.

els) for collision detection, and a build tile map (with a resolution of  $32 \times 32$  pixels) for determining where new buildings can be placed. In our case we decided to use build tile resolution for influence maps since all units are bigger than  $8 \times 8$  pixel size, and thus this resolution is enough for kiting.

For the purposes of kiting, each unit  $u_1$  performing kiting will compute an influence map. We are interested only on storing two pieces of information in our influence maps: whether a cell is inside of the range of fire of an enemy, and whether a cell is too close to a wall and a unit might run the risk of getting trapped. Those two pieces of information are stored in the influence map in the following way.

**Enemy units:** We assigned an influence field to each enemy  $u_2$  with a constant influence value based on the distance in Eq. 2. Then we use Eq. 3 to define the maximum distance  $d_{max}(u_1, u_2)$  of the field on Eq. 4:

$$d_{max}(u_1, u_2) = u_2.attackRange + k + u_2.speed \times kitingTime(u_1) \quad (3)$$

Where  $k$  is a confidence constant value to ensure a safe distance. In our case we established this constant with a value of 1. The value added to the influence map to a position at distance  $d$  from an enemy  $u_2$  is defined as:

$$I_{enemy}(u_1, u_2, d) = \begin{cases} u_2.DPS & \text{if } d \leq d_{max}(u_1, u_2) \\ 0 & \text{if } d > d_{max}(u_1, u_2) \end{cases} \quad (4)$$

**Walls:** One of the problems on a kiting movement is getting stuck on a corner or in a closed area. In order to avoid this, we define an influence field on each wall.

$$I_{wall}(d) = \begin{cases} 1 & \text{if } d \leq 3 \\ 0 & \text{if } d > 3 \end{cases} \quad (5)$$

Where 3 is the radius of the field for walls, which was determined empirically.

We can see an example of generated Influence Map on Figure 3. The influence map defined in this section can be used by a unit to flee while performing kiting. The next section describes how to decide which of the enemy units to attack when performing kiting.

## Target Selection

Selecting the right target is essential to maximize the effectiveness of kiting. In our approach, units select a target by assigning a score to each enemy unit, and then selecting the unit with the maximum score. The score is based on three factors: distance to target, tactical threat and aggro<sup>1</sup>.

**Distance:** the pixel distance to the target.

**Tactical threat:** a fixed, hand assigned, value to each different unit type depending on its features and abilities.

**Aggro:** the result of the DPS (Damage Per Second) that would be inflicted to the target by the attacking unit divided by the time to kill the target.

We can calculate the *aggro* of unit  $u_1$  for unit  $u_2$  in the following way:

$$aggro(u_1, u_2) = \frac{u_2.DPS \text{ to } u_1}{\text{time for } u_1 \text{ to kill } u_2} \quad (6)$$

$$\text{time for } u_1 \text{ to kill } u_2 = \frac{u_2.HP}{u_1.DPS \text{ to } u_2} \quad (7)$$

The score of an enemy unit is computed as the weighted sum of the three factors:

$$Score(u) = aggro(u) \times w_1 + tactical(u) \times w_2 + distance(u) \times w_3 \quad (8)$$

By default NOVA gives a lot of importance to aggro since this variable indicates the most dangerous units that you can kill more quickly. But this can lead in a wrong target selection on kiting behavior; skipping the closest units to attack the one with the highest score, exposing our unit to enemy fire as a consequence.

To solve this issue in order to successfully perform kiting, the vale of  $w_3$  needs to be high enough to make the distance the most significant factor.

## Kiting Algorithm

Figure 4 shows the kiting movement algorithm, where *targetSelection* returns the best available target for the current unit using Eq. 8; *canKite* uses Eqs. 1 and 2 to check if the current unit can kite the enemy unit selected; and *getSecurePosition* gets the unit's actual position and returns the closest secure position on the Influence Map. Then if we are in a secure position we can attack the target, otherwise the unit must keep fleeing.

In practice, it is not necessary to create a different influence map for each friendly unit reforming kiting, since all units of the same type can share the same influence map.

## Empirical Evaluation

In order to evaluate our approach, we performed three sets of experiments in the domain of StarCraft. We incorporated our

<sup>1</sup>“aggro” is a slang term meaning “aggravation” or “aggression”. In RTS and role-playing games, aggro denotes the aggressive interests of a unit.

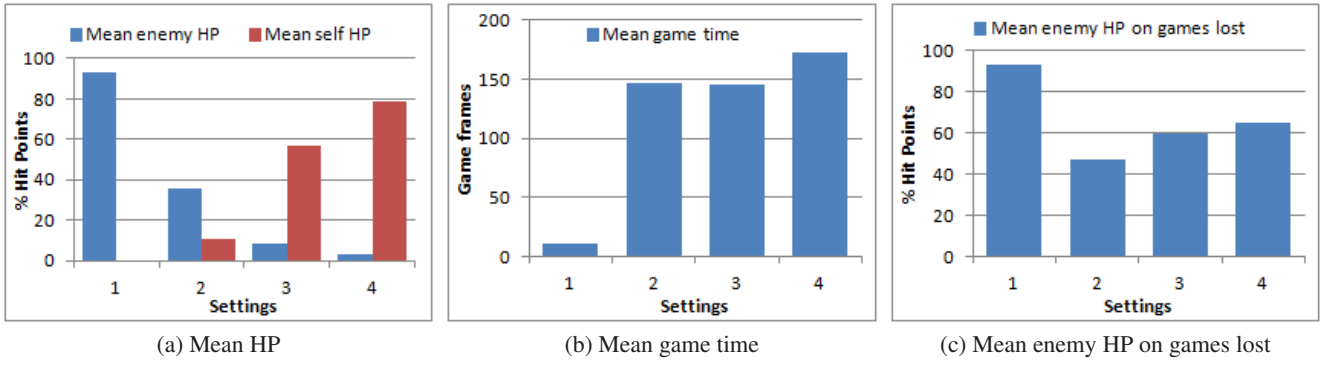


Figure 5: Experiment 1: Results for each of the different settings.

```

tick() {
    target = targetSelection();
    if (canKite(target)) {
        kitingAttack(target);
    } else {
        attack(target);
    }
}

kitingAttack(target) {
    position = getSecurePosition(actualPos);
    if (position == actualPos) {
        attack(target);
    } else {
        move(position); // flee movement
    }
}

```

Figure 4: High level algorithm to perform kiting.

approach into NOVA, and we confronted it against the built-in AI of StarCraft. Kiting was incorporated into the behavior that controls the Terran Vulture units of NOVA, since they are good units to perform kiting.

The first two experiments evaluate kiting in isolation by pitting Vultures against groups of opponent units. The third experiment evaluates the impact of kiting in the overall performance of NOVA by doing full-game experiments.

### Experiment 1: One-Unit Kiting

In this experiment we show the performance increment that each influence field and the improved target selection brings. We run the experiments in an open square map with one Vulture (our unit) against six Zealots (controlled by the built-in StarCraft AI).

We configured NOVA in 4 different ways in order to evaluate all the different parts of our kiting approach:

**Settings 1:** Default behavior. Here we use the default attack movement of NOVA, that means that when we order a unit to attack a target, that unit will keep attacking without any flee movement.

**Settings 2:** Influence Map (Enemy field). In this case we

Settings	1	2	3	4
Games won on Experiment 1	0.0%	24.9%	85.5%	95.2%
Games won on Experiment 2	0.0%	98.8%	100.0%	100.0%
Games won on Experiment 3	17.6%	-	-	96.0%

Table 2: Win ratio on each experiment and setting

execute a kiting movement only assigning charges on enemy units.

**Settings 3:** Influence Map (Enemy and Wall fields). Here we execute a kiting movement assigning charges on enemy units and map walls.

**Settings 4:** IM and target selection. Same as previous scenario but now we use our proposed target selection method. This is the full kiting behavior approach presented in this paper.

Our hypothesis is that each successive configuration is better than the previous one. Table 2 shows the win ratio of NOVA controlling only one Vulture against four Zealots after executing the experiment 1000 times on each setting, showing that kiting drastically increases the performance of NOVA, from losing all the games, to winning 95.2% of the time.

Figure 5a shows a more detailed analysis of the obtained results by plotting the hit points ratio of the remaining units at the end of a game. As expected, in each configuration our final HP is higher than in the previous one. However, if we look at the enemy HP only considering the games we lost (Figure 5c) its HP also goes up. The main reason of that is because on each configuration, NOVA is more cautious than in the previous one since units spend more time fleeing than attacking. This is also reflected on game time. If we look at Figure 5b we can observe how by using kiting, games take a longer time, since NOVA's units spend more time fleeing to avoid enemy fire.

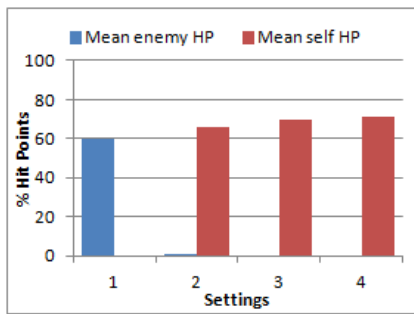


Figure 6: Experiment 2: Mean % HP at the end of the game.

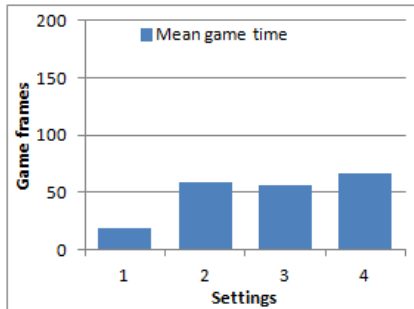


Figure 7: Experiment 2: Mean game time.

### Experiment 2: Group Scalability

In this experiment we want to test how does our kiting behavior scale to groups of units and test whether the two fields we defined are enough. To this end, we performed the same experiments as before but in a map where NOVA controls four Vultures against six Zealots; the results are shown in Table 2. We observed that units did not disturb each other, as can be seen by the win ratios in Table 2. Our kiting approach was able to win 100.0% of the games.

Figure 6 shows that the HP at the end of the game are very high even for simpler settings of NOVA. Finally, as expected, the time required to win a game is dramatically reduced compared to Experiment 1 as shown on Figure 7.

### Experiment 3: Full Game

Until now, we tested our approach in a small melee scenario. This set of experiments aim at evaluating the utility of kiting in a full game. For this reason we evaluated the performance of NOVA against the built-in Protoss AI of Starcraft. We used two different settings of NOVA:

**Settings 1:** Default behavior. Here we use the default attack movement of NOVA.

**Settings 4:** Kiting behavior. In this case NOVA uses a kiting behavior when possible.

Table 2 shows the average of 1000 games with each configuration. The result is that the percentage of victories increases 445.45%, showing that kiting has a huge impact in the performance of NOVA, by helping its units survive for longer when fighting against enemy units against which kiting can be performed.

## Conclusions

In this paper we have studied the implementation of tactical movement for real-time strategy games, which is part of our long term goal of developing AI techniques capable of handling large scale real-time domains. Specifically, in this paper we have presented and evaluated an approach to perform kiting behavior using Influence Maps. Our experiments show the impact that incorporating effective tactical moves like kiting can have in the performance of complete RTS game playing bots, such as NOVA. Additionally, our approach to kiting is computationally tractable, and can be used in real-time bots. As a proof, we incorporated this approach into the NOVA bot, which participated in the annual StarCraft AI Competition (Weber 2010).

As part of our future work, we would like to build on top of our approach and explore more complex tactical moves. For instance, using kiting to lead the enemy towards a certain part of the map for ambushing. Some more complex moves might require explicit unit cooperation by, for instance, adding friendly-unit charges onto the Influence Map.

## References

- Avery, P.; Louis, S. J.; and Avery, B. 2009. Evolving coordinated spatial tactics for autonomous entities using influence maps. In *CIG*, 341–348.
- Buro, M. 2003. Real-time strategy games: A new ai research challenge. *International Joint Conference on Artificial Intelligence* 1534–1535.
- Hagelbäck, J., and Johansson, S. J. 2008. The rise of potential fields in real time strategy bots. In Darken, C., and Mateas, M., eds., *Proceedings of AIIDE 2008*.
- Johansson, S. J., and Saffiotti, A. 2002. Using the electric field approach in the robocup domain. In *RoboCup 2001: Robot Soccer World Cup V. Volume 2377 of Lecture Notes in Artificial Intelligence*, 399–404. Springer.
- Khatib, O. 1985. The potential field approach and operational space formulation in robot control. In *Proc. of the 4th Yale Workshop on Applications of Adaptive Systems Theory*.
- Liu, L., and Li, L. 2008. Regional cooperative multi-agent q-learning based on potential field. 535–539.
- Preuss, M.; Beume, N.; Danielsiek, H.; Hein, T.; Naujoks, B.; Piatkowski, N.; Str, R.; Thom, A.; and Wessing, S. 2010. Towards intelligent team composition and maneuvering in real-time strategy games. *IEEE Trans. Comput. Intellig. and AI in Games* 2:82–98.
- Reynolds, C. W. 1999. Steering behaviors for autonomous characters. *Proceedings of Game Developers Conference 1999* 763–782.
- Thurau, C.; Bauckhage, C.; and Sagerer, G. 2000. Learning human-like movement behavior for computer games.
- Tozour, P. 2001. Influence mapping. In DeLoura, M., ed., *Game Programming Gems 2*. Charles River Media. 287–297.
- Uriarte, A. 2011. Multi-reactive planning for real-time strategy games. Master’s thesis, UAB.
- Weber, B. 2010. AIIDE 2010 StarCraft competition.