Special Issue Short Paper

Hybrid Pathfinding in *StarCraft*

Johan Hagelbäck

***Abstract*—Micromanagement is a very important aspect of real-time strategy (RTS) games. It involves moving single units or groups of units effectively on the battle field, targeting the most threatening enemy units and use the unit's special abilities when they are the most harmful for the enemy or the most beneficial for the player. Designing good micromanagement is a challenging task for AI bot developers. In this paper, we address the micromanagement subtask of positioning units effectively in combat situations. Two different approaches are evaluated, one based on potential fields and the other based on flocking algorithms. The results show that both the potential fields version and the flocking version clearly increases the win percentage of the bot, but the difference in wins between the two is minimal. The results also show that the more flexible potential fields technique requires much more hardware resources than the more simple flocking technique.**

***Index Terms*—Autonomous agents, software agents.**

I. Introduction

Real-time strategy (RTS) games provide many challenges for AI bot developers. Each player typically starts with a command center and a number of workers. The workers are used to gather one or more type of resources, which in turn is used to construct buildings. Buildings can be used to expand the base, protect the base, and produce combat units. It is also common for RTS games to have technology trees where a player can invest resources in upgrades for units and/or buildings. Each player must take numerous decisions about what to spend resources on. Resources are limited and take time to gather. They can, for example, be spent on cheap units to be able to launch an attack on the enemy early in a game (often referred to as a rush tactic), constructing a good defence with nonmobile turrets and bunkers, or on technical advancements to produce strong units, which on the contrary can leave the player vulnerable early in the game.

Decisions in RTS games or usually divided into the two categories macro- and micromanagement. Macromanagement is decisions at a higher level of abstraction such as what buildings and units to produce, which upgrades to invest in, and when to launch an attack on enemy bases. Micromanagement is about controlling individual units. Placing units at tactically good locations, targeting the most dangerous enemy units within fire range and smart use of special abilities can win many games. This paper deals with micromanagement, specifically where to move individual units in combat situations. Examples of good placement of units is to place long-range units in the back guarded by short-range units in the front, avoid taking unnecessary damage on support units, and surround the enemy at maximum shooting distance.

Navigation of units in RTS games is typically handled with pathfinding algorithms such as A*. A* always finds the best possible path between two positions in a reasonably short time, but does not handle dynamic worlds (such as in RTS games) very well. It takes time for a unit to travel along a path and much can happen in the game world before the unit reaches the goal position. If the path is suddenly blocked by a mobile object it becomes obsolete and the agent has to recalculate all or parts of it. Extensive work has been done to modify A* to work better in highly dynamic worlds. Silver proposes an addition of an extra time dimension to the pathfinding graph to allow units to reserve a node at a certain time [1]. The work of Olsson addresses the issue of changes in the pathfinding graph due to the construction or destruction of buildings [2]. Koenig and Likachev have made contributions to the field with their work on real-time A* [3], [4]. Pathfinding algorithms such as A* only find paths to the goal position and does not take into account how to position units in combat situations. Additional logic has to be added to handle positioning of units.

In a previous paper, we have shown that positioning of units in combat situations can be achieved by using potential fields based solutions [5]. The paper proposes a hybrid approach for navigation where A* is used when no enemy units or buildings are within sight range, and potential fields when unit(s) are engaged in combat. The hybrid approach avoids the problem of local optima when using potential fields (units can get stuck in complex terrain) by using A* while at the same time getting the benefits of potential fields for positioning of units in combat situations.

The purpose of this paper is to evaluate if the potential fields based part of the hybrid navigation system can be replaced with a system based on flocking algorithms.

Boids is a flocking algorithm for simulating the aggregate motion of a flock of birds, a herd of land animals or a school of fish. It was first published by Reynolds in 1987 [6] and has since then attracted a lot of attention in different application areas. The boids algorithm is based on a set of rules. In the simplest version, three rules are used.

- Separation: Each member of the flock moves in a direction to avoid colliding with local flock mates.
- Alignment: Each member of the flock moves towards the average heading of the flock members.
- Cohesion: Each member of the flock moves towards the average position of the flock members.

Manuscript received July 29, 2014; revised December 16, 2014, February 19, 2015; accepted March 14, 2015. Date of publication March 18, 2015; date of current version December 13, 2016.

The author is with the Department of Computer Science, Linnaeus University, Växjö 35195, Sweden (e-mail: johan.hagelback@lnu.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2015.2414447

Additional rules can be added for, for example, moving towards a goal and avoiding obstacles.

Flocking have previously been used with success in RTS games. Danielsiek *et al.* used boids in the open source RTS game Glest [7]. The authors combined boids with influence maps to find safer paths for the flocks. The authors show that flocking decreased the amount of unit losses. Preuss *et al.* extended the work in [8]. Synnaeve and Bessier successfully used flocking combined with a Bayesian model for unit control in the RTS game *StarCraft* [9].

The hybrid A*/boids navigation system has been implemented in the open-source *StarCraft* bot BTHAI.[1]The bot is a slightly updated version of the one used in our previous work where a hybrid A*/potential fields navigation system was used [5].

II. Hybrid Navigation System

The hybrid navigation system has two parts. When no enemy unit or building is within sight range, agents navigate using A*. Local influence algorithms such as flocking and potential fields have a tendency to get stuck in complex terrain since they do not "backtrack" if they get stuck in a dead end. We avoid this problem by using A* to calculate the shortest path to the goal position.

A* is on the other hand not very suitable for positioning units. Agents move towards the goal without considering how to effectively engage the enemy in a combat situation. To solve this the navigation system switches to using flocking with the boids algorithm as soon as an enemy unit or building is within sight range. The boids algorithm is modified so agents try to keep a distance to the enemy close to the maximum shooting distance of its own weapons, while at the same time keeping the squad (the group of units an agent belongs to) grouped up. Agents should also avoid colliding with other own agents and obstacles.

The hybrid navigation system is outlined in Fig. 1.

Fig. 2 shows the preferred moves of two Terran Marines in a combat situation where four Marines attack a Protoss building. Two Marines are already located at maximum shooting distance, one is too far away and the last one is too close to the enemy. The Marine that is too far away shall move to a spot on the maximum shooting distance without colliding with other own units. The same goes for the Marine that is too close, but note that shooting has higher priority than moving so he only moves when his weapon is on cooldown.

Fig. 3 shows how this behavior looks like in-game. A group of marines and medics group up at maximum shooting distance and form an arc surrounding the enemy buildings.

The hybrid potential fields version works in a similar fashion. Pathfinding with A* is used when no enemy units are within sight range, but potential fields is used instead of boids when an agent is close to an enemy.

In potential fields, each interesting game object generates a field surrounding the object. Each field can vary in shape and size, but must fade to zero. Positive potential field values are attracting for agents, negative values are repelling and zero has no influence. All fields generated by objects are weighted and summed to a total potential field, which is used by agents for navigation. When deciding where to move, each agent locates

[1]BTHAI Project—http://code.google.com/p/bthai/

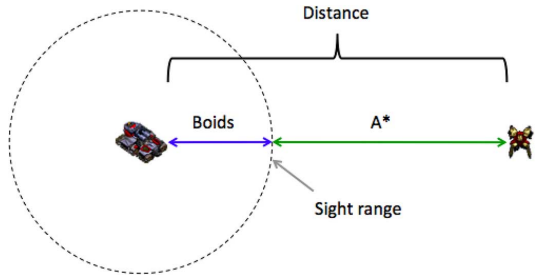

Fig. 1. Hybrid navigation system.

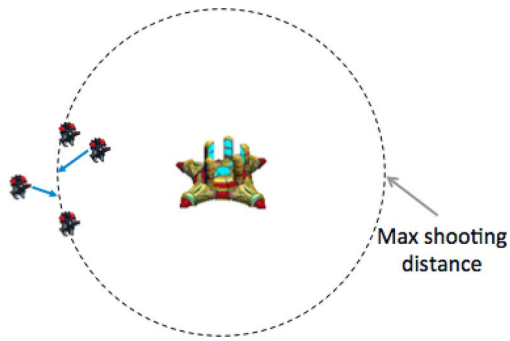

Fig. 2. Example of preferred moves for two agents.

and moves towards the most attracting position in its near surroundings. An example of how a total potential field could look like is shown in Fig. 4. The figure shows three agents (green circles) that generate small repelling fields around them for obstacle avoidance, impassable terrain (brown) that also generate repelling fields for obstacle avoidance, and an enemy unit (red circle) that is attracting for the agents. The highest potential values (light blue areas) are located at the maximum shooting distance from the enemy unit. This has the effect that the agents approach and surround the enemy units at the shooting distance of their weapons.

Potential fields can also be used when retreating from dangerous enemy units. In this case, an enemy unit generates a repelling field of negative values with a size a little larger than the maximum shooting distance of the enemy unit. An example of agents retreating from an enemy unit is shown in Fig. 5. The repelling field is shown in red where lighter red areas are more repelling than darker red areas.

Fig. 6 shows an example of the generated total potential field surrounding a Terran Marine in *StarCraft*. As in the previous examples the most attracting areas (light blue) are at the maximum shooting distance of the Terran Marine. More details about the potential field implementation can be found in a previous paper [5].

III. Boids Implementation

All agents are grouped in squads. A squad can contain any number of units and can have different unit types. Note that the boids rules described in this chapter only applies to group members in an agent's own squad. The rules do not take agents from other squads into consideration. The implementation has five rules, each described in detail below. The pseudocode for each rule, except the separation from enemy units rule, contains a constant value. This value determines how dominant each rule is, and is in our case simply set by trial-and-error.

Fig. 3. Screenshot from the boids bot version.

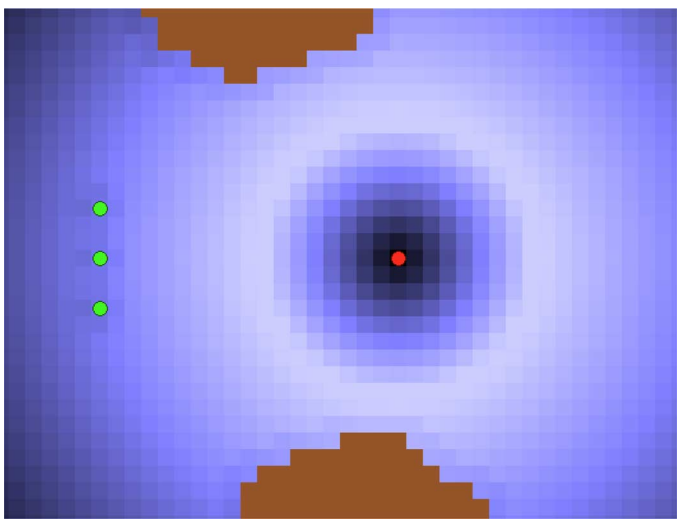

Fig. 4. Example of generated potential field for three own units (green) approaching an enemy unit (red).

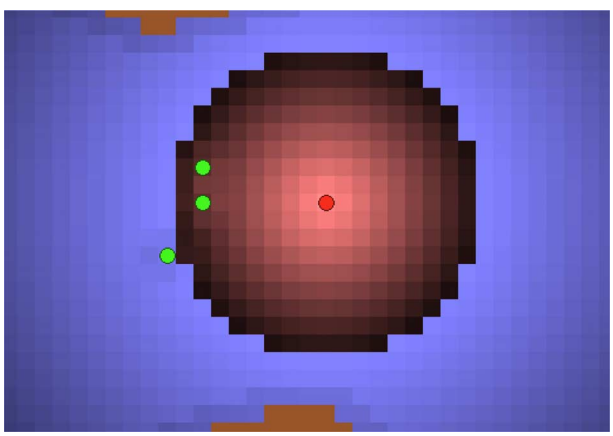

Fig. 5. Example of generated potential field for three own units (green) retreating from an enemy unit (red).

A. Cohesion

It is important for the members of a squad to stay together. If single agents get astray they are easy prey for the enemy. This is solved with the cohesion rule—each agent moves towards the average position of all other members in the squad (see pseudocode later). This rule is not very dominant since other rules, such as keeping distance to enemy units, are more important.

```
function ApplyCohesion(Agent a)
{
  foreach (Agent o in squad.Members())
  {

if (o.ID != a.ID)
      {
        dX+ = o.Position().x();
        dY+ = o.Position().y();
      }
                          }
  dX = dX / (squad.Size() -1);
  dY = dY/ (squad.Size() -1);
  a.dX+ = (dX - a.Position().x())/100;
  a.dY+ = (dY - a.Position().y()) / 100;
}
```

B. Alignment

Each squad move towards a common goal and therefore individual agents shall move in approximately the same direction. Each agent therefore moves towards the average heading/direction of the other squad members. Pseudocode for the alignment rule is shown later. The influence of this rule is quite weak as in the case of the cohesion rule.

```
function ApplyAlignment(Agent a)
{
  foreach (Agent o in squad.Members())
  {
    if (o.ID != a.ID)
    {
      //Heading is in radians
      dX+ = cos(o.Heading());
      dY+ = sin(o.Heading());
    }
  }
  dX = dX/ (squad.Size() -1);
  dY = dY/ (squad.Size() -1);
  a.dX+ = (dX - cos(a.Heading()) / 5;
  a.dY+ = (dY - sin(a.Heading()) / 5;
}
```

C. Goal

The bot has a commander that decides where to move each squad. The commander controls one or more attacking squads. When all these squads are constructed (all squad members have been built) an attack is launched at the enemy. The goal of the attack is the closest enemy building found during scouting. The commander is very simple and a lot of features and improvements can be made, but that is out of scope of this paper. Each agent moves towards the goal of the squad (pseudocode below). Note that this rule has quite weak influence as well. The navigation towards the goal is mostly handled by the A* part of the navigation system.

```
function ApplyGoal(Agent a)
{
  int goalX = squad.Goal().x();
  int goalY = squad.Goal().y();
  a.dX+= (goalX - a.Position().x()) / 100;
  a.dY+= (goalY - a.Position().y()) / 100;
}
```

D. Separation—Own Agents

Agents shall avoid colliding with other own agents and shall also strive to keep a short distance between each other. This is handled with the separation rule. The detection limit (the longest distance for which the rule has any influence) is calculated as the radius of the agent plus the radius of the agent to avoid plus two. If we want the group members to spread out more, we simply increase the detection limit. Pseudocode for the separation of own agents is shown here.

```
function ApplySeparationAgents(Agent a)
{
  foreach (Agent o in squad.Members())
  {
    float detectionLimit = a.Radius() + o.Radius()
+ 2;
    if (o.ID! = a.ID)
    {
      float distance = a.DistanceTo(o);
      if (distance <= detectionLimit)
      {
        dX− = (o.Position().x() -
a.Position().x());
        dY− = (o.Position().y()
-a.Position().y());
      }
    }
  }
  a.dX+ = dX/ 5;
  a.dY+ = dY/5;
}
```

E. Separation—Enemy Units

The separation from enemy units is the most important rule in the boids implementation. The purpose of the rule is to keep agents at approximately the maximum shooting distance from enemy units and buildings. The detection limit is calculated differently depending on the current situation.

- If the agent can attack the enemy, detection limit is set to the maximum shooting distance of the agent's weapon.
- If the agent cannot attack the enemy but the enemy can attack the agent, detection limit is set to the maximum shooting distance of the enemy's weapon plus the radius of the agent.
- If the agent cannot attack the enemy and the enemy cannot attack the agent, detection limit is set to the radius of the agent plus 10.

The pseudocode for the separation from enemy units rule is as follows.

```
function ApplySeparationEnemy(Agent a)
{
  foreach (Unit e in Enemies())
  {
    float detectionLimit = a.ShootDist(e);
  if (a.IsSupport())
    {
      detectionLimit = e.ShootDist(a) +5;
    }
    float distance = a.DistanceTo(e);
    if (distance <= detectionLimit)
    {
      dX− = (e.Position().x() -
a.Position().x());
      dY− = (e.Position().y()
-a.Position().y());
    }
  }
  a.dX+ = dX;
a.dY+ = dY;
}
```

F. Separation—Terrain

Agents shall also avoid colliding with impassable terrain. This separation rule is explained in the pseudocode below. Note that this rule does not apply to airborne units. There is no terrain in *StarCraft* that blocks flying units.

```
function ApplySeparationTerrain(Agent a)
{
  WalkTile aWT = a.WalkTilePosition();
  foreach (WalkTile tWT in aWT.Adjacent())
  {
      if (!tWT.IsWalkable())
      {
        float detectionLimit = a.Radius() +
tWT.Width()/2;
        if (distance <= detectionLimit)
        {
          dX− = (tWT.Position().x() -
a.Position().x());
          dY− = (tWT.Position().y()
-a.Position().y());
      }
  }
  a.dX+ = dX/10;
  a.dY+ = dY/10;
}
```

IV. Experiments

The main purpose of this paper is to compare two hybrid pathfinding systems, one based on boids and another based on potential fields. We will also see how effective the hybrid pathfinding systems are compared to a nonhybrid pathfinding system based on A* only. One of the disadvantages of potential

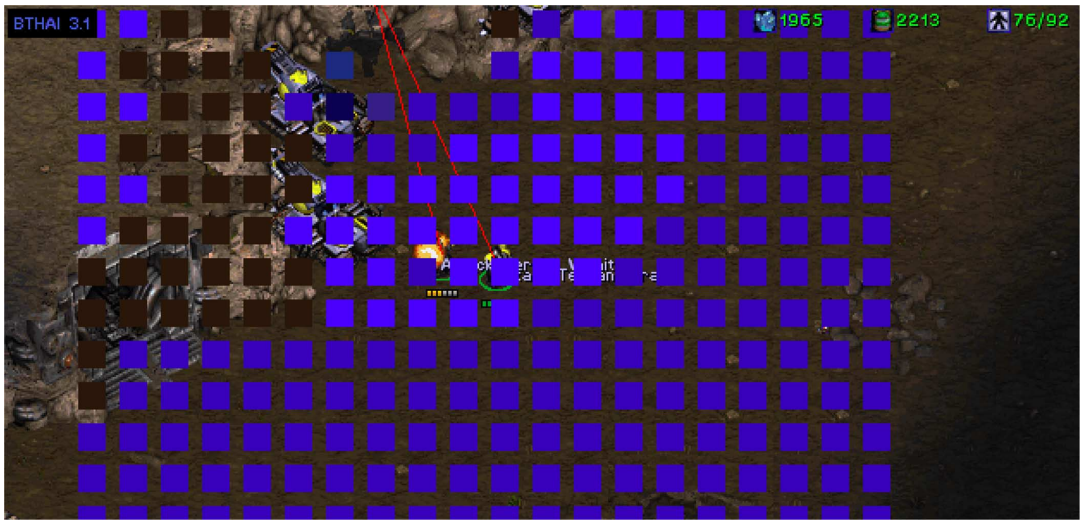

Fig. 6. Screenshot from the potential fields bot version.

fields based solutions in games with many agents (such as RTS games) is performance, and a performance comparison between potential fields and boids would give valuable insight in which hybrid pathfinding solution to select.

The following research questions will be addressed.

- Is a hybrid pathfinding system better in terms of win ratio compared to a nonhybrid pathfinding system?
- Is a hybrid pathfinding system based on boids as good as a system based on potential fields in terms of win ratio?
- How much less computational resources are needed for a hybrid pathfinding system based on boids compared to a system based on potential fields?

To answer the research questions we have conducted a series of experiments where the three pathfinding systems (hybrid boids, hybrid potential fields, and nonhybrid) played against the built-in *StarCraft* AI on four *StarCraft* maps. Both the bot and the built-in AI played Terrans, and each map was played 50 times for each bot version. The following maps were used.

- Fading realm—a simple two-player map the bot usually performs very well at.
- Destination—another two-player map the bot often performs very well at.
- Fortress—a larger four-player map we know the bot has some difficulties with.
- Empire of the sun—a four-player map that is slightly easier for the bot compared to fortress.

We have also compared performance between the boids hybrid pathfinder and the potential fields hybrid pathfinder using QueryPerformanceCounter [10], a reliable high-precision timer that can measure down to ms level. The performance tests were conducted on an Intel Core i5–4200 M CPU with 8 GB of RAM and Windows 8.1 64 bits.

V. Results

The results from the experiments are presented in Table I. Not surprisingly, a hybrid pathfinder using either potential fields or boids outperformed the version without any hybrid pathfinding. The difference between the two hybrid pathfinding systems was minimal, 175 wins (potential fields) against 178 wins (boids). There is also a clear difference between simple two-player maps and more complex four-player maps. The results for hybrid and nonhybrid pathfinding was almost equal for the two-player maps, while the results differ markedly for the four-player maps.

TABLE I
Results From the Experiments. The Percentages Are Win Ratios Against the Built-in AI

Map	Games	Wins (A*)	Wins (PF)	Wins (Boids)
Fading Realm	50	96%	98%	98%
Destination	50	98%	100%	100%
Fortress	50	50%	66%	68%
Empire o t Sun	50	72%	86%	88%
Total	200	79%	88%	89%

TABLE II
Results from Pairwise Two-Sample T-Test Between Proportions

Pair	P-value	Significant
A* - PF	0.01	Yes
A* - Boids	0.00	Yes
PF - Boids	0.38	No

To test for statistical significance we have performed pairwise two-sample T-test between proportions with significance level 0.10. The results are presented in Table II. As expected the win percentage of A* is significantly lower than the other two versions, but no significant difference can be found between the potential fields and boids solutions.

The experiments failed to find a difference in win percentage between the potential fields and the boids solutions for hybrid pathfinding. This is as we expected. The behavior of squads is quite similar when watching games being played.

Since both versions are equal in terms of winning games, is there a difference in how much hardware resources each of them requires? The results from the performance tests are presented in Table III. The boids version had an average execution time of 0.063 ms over five games and 17 396 method calls. The potential fields version on the other hand had an average execution time of 7.732 ms over five games and 13 518 method calls. The potential

TABLE III
RESULTS FROM THE PERFORMANCE TESTS

Version	Avg exec time	Total calls	Max exec time
PF	7.632 ms	13518	20.337 ms
Boids	0.063 ms	17396	0.625 ms

TABLE IV
RESULTS FROM THE BOIDS VERSION PLAYING AGAINST THE POTENTIAL FIELDS VERSION

Map	Games	Wins (PF)	Wins (Boids)
Fading Realm	10	60%	40%
Destination	10	60%	40%
Fortress	10	50%	50%
Empire o t Sun	10	50%	50%
Total	40	55%	45%

fields version required on average well over 100 times longer to execute! Of course there are probably several ways the potential fields implementation can be optimized, but the results at least give a good hint that boids is a much faster solution.

To see which hybrid pathfinding system is the best in terms of win ratio, we have also run a series of experiments where the boids version played against the potential fields version. The results are presented in Table IV. The potential fields based version has a slightly higher win ratio, but at the cost of higher computational needs.

VI. CONCLUSION AND DISCUSSION

We expected that a hybrid pathfinding system would outperform a nonhybrid solution. This was verified by the experiments, but an interesting note is that the difference in win ratio is very small for simple two-player maps and quite large for more complex four-player maps. Overall we think that hybrid pathfinders are very competent for RTS game bots. This claim is also verified in a previous paper [5].

We also expected the win ratio of the boids solution to be equal or slightly lower than for the potential fields solution. This was also verified by the experiments. Even though the boids solution had slightly higher win ratio (89%) compared to the potential fields solution (88%) the difference was not statistically significant and we conclude that the win ratios are in practice equal.

The most interesting thing is the performance tests. We expected that the potential fields solution would require much more execution time than a solution based on boids. This was also verified by the experiment results. The difference in execution time between the two was well above the factor of 100. The potential fields based solution in its current form is not suitable for *StarCraft* since the average execution time of 7.6 ms (with a peak at 20.3 ms) is too long. The AI has to take a lot of micro- and macrodecisions in the relatively short time available each frame, and currently the bot cannot run at highest speed when using potential fields. More research is required to optimize implementations of potential fields based solutions.

An important benefit of potential fields over boids is that fields surrounding in-game objects can have a wide variety of shapes and sizes. A field can for example be attractive at one distance from the object and at the same time repelling at another distance. Boids is not as flexible. In-game objects are either attractive or repelling. This is to some extent verified in the second experiment where the potential fields based solution outperformed the, compared to the built-in AI, more competent boids solution when playing against each other on small maps. On larger four-player maps both versions had the same win ratio. We believe this is due to more factors influencing who will win or lose on a larger map (for example luck with the randomly selected starting positions) than on a smaller map. A potential fields based solution is likely to be more effective than a boids solution when facing better opponents that also tries to surround the enemy. As future work both versions could be played against bots from the annual *StarCraft* tournaments to see if this is the case. Developers should also have in mind that it can be quite time consuming to fine tune how individual potential fields interact to form a total potential field for effective behaviors.

The choice of which technique to use comes down to the complexity of the agent behaviors. We believe that for hybrid pathfinding in RTS games where the goal of the agents is to surround the enemy, a boids solution can be at least as effective as a potential fields based solution while requiring much less hardware resources. Note that more research has to be done to see if this is also the case when facing more competent opponents than the built-in AI.

An interesting idea for future work is to combine flocking with potential fields. Potential fields can be used for tactical analysis of combat situations (similar to combining flocking with influence maps as proposed by Danielsiek *et al.* [7]) and boids for agent movements.

REFERENCES

[1] D. Silver, "Cooperative pathfinding," in *AI Game Programming Wisdom 3*. Newton Center, MA, USA: Charles River Media, 2006.
[2] P.-M. Olsson, "Practical pathfinding in dynamic environments," in *AI Game Programming Wisdom 4*. Newton Center, MA, USA: Charles River Media, 2008.
[3] S. Koenig, "A comparison of fast search real-time situated agents," in *Proc. Autonom. Agents Multi-Agent Syst. (AAMAS)*, 2004.
[4] S. Koenig and M. Likhachev, "Real-time adaptive A*," in *Proc. Autonom. Agents Multi-Agent Syst. (AAMAS)*, 2006.
[5] J. Hagelbäck, "Potential-field based navigation in StarCraft," in *Proc. IEEE Conf. Computat. Intell. Games (CIG)*, 2012.
[6] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," in *Proc. Comput. Graph. (SIGGRAPH'87)*, 1987, vol. 21, no. 4, pp. 25–34.
[7] H. Danielsiek *et al.*, "Intelligent moving of groups in real-time strategy games," in *Proc. IEEE Symp. Comput. Intell. Games (CIG)*, 2008.
[8] M. Preuss *et al.*, "Towards intelligent team composition and maneuvering in real-time strategy games," *IEEE Trans. Comput. Intell. AI Games*, pp. 82–98, 2010.
[9] G. Synnaeve and P. Bessiere, "A Bayesian model for RTS units control applied to StarCraft," in *Proc. IEEE Symp. Computat. Intell. Games (CIG)*, 2011.
[10] "QueryPerformanceCounter function," 2014 [Online]. Available: msdn.microsoft.com/en-us/library/windows/desktop/ms644904(v=vs.85).aspx