Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 910819, 10 pages
doi:10.1155/2009/910819

Research Article

A Multiagent Potential Field-Based Bot for Real-Time Strategy Games

Johan Hagelbäck and Stefan J. Johansson

Department of Software and Systems Engineering, Blekinge Institute of Technology, P.O. Box 520, 372 25 Ronneby, Sweden

Correspondence should be addressed to Johan Hagelbäck, jhg@bth.se

Received 30 April 2008; Accepted 7 September 2008

Recommended by Abdennour El Rhalibi

Bots for real-time strategy (RTS) games may be very challenging to implement. A bot controls a number of units that will have to navigate in a partially unknown environment, while at the same time avoid each other, search for enemies, and coordinate attacks to fight them down. Potential fields are a technique originating from the area of robotics where it is used in controlling the navigation of robots in dynamic environments. Although attempts have been made to transfer the technology to the gaming sector, assumed problems with efficiency and high costs for implementation have made the industry reluctant to adopt it. We present a multiagent potential field-based bot architecture that is evaluated in two different real-time strategy game settings and compare them, both in terms of performance, and in terms of softer attributes such as configurability with other state-of-the-art solutions. We show that the solution is a highly configurable bot that can match the performance standards of traditional RTS bots. Furthermore, we show that our approach deals with Fog of War (imperfect information about the opponent units) surprisingly well. We also show that a multiagent potential field-based bot is highly competitive in a resource gathering scenario.

1. Introduction

A *real-time strategy* (RTS) game is a game in which the players use resource gathering, base building, technological development and unit control in order to defeat its opponent(s), typically in some kind of war setting. The RTS game is not turn-based in contrast to board games such as Risk and Diplomacy. Instead, all decisions by all players have to be made in real time. Generally the player has a top-down perspective on the battlefield although some 3D RTS games allow different camera angles. The real-time aspect makes the RTS genre suitable for multiplayer games since it allows players to interact with the game independently of each other and does not let them wait for someone else to finish a turn.

In RTS games computer bots often "cheats," that is, they have complete visibility (perfect information) of the whole game world. The purpose is to have as much information available as possible for the artificial intillegence (AI) to reason about tactics and strategies in a certain environment. Cheating is, according to Nareyek, "*very annoying for the player if discovered*" and he predicts the game AIs to get a larger share of the processing power in the future which in turn may open up for the possibility to use more sophisticated AIs [1]. The human player in most modern RTS games does not have this luxury, instead the player only has visibility of the area populated by the own units, and the rest of the game world is unknown until it gets explored. This property of incomplete information is usually referred to as *Fog of War* or FoW.

In 1985, Ossama Khatib introduced a new concept while he was looking for a real-time obstacle avoidance approach for manipulators and mobile robots. The technique which he called *Artificial Potential Fields* moves a manipulator in a field of forces. The position to be reached is an attractive pole for the end effector (e.g., a robot) and obstacles are repulsive surfaces for the manipulator parts [2]. Later on Arkin [3] updated the knowledge by creating another technique using superposition of spatial vector fields in order to generate behaviors in his so called motor schema concept.

Many studies concerning potential fields are related to spatial navigation and obstacle avoidance (see, e.g., [4, 5]). The technique is really helpful for the avoidance of simple

obstacles even though they are numerous. Combined with an autonomous navigation approach, the result is even better, being able to surpass highly complicated obstacles [6].

Lately some other interesting applications for potential fields have been presented. The use of potential fields in architectures of multi agent systems is giving quite good results defining the way of how the agents interact. Howard et al. developed a mobile sensor network deployment using potential fields [7], and potential fields have been used in robot soccer [8, 9]. Thurau et al. [10] have developed a game bot which learns reactive behaviours (or potential fields) for actions in the first-Person Shooter game Quake II through imitation.

The article is organised as follows. First, we propose a methodology for multiagent potential field- (MAPFs-) based solution in an RTS game environment. We will show how the methodology can be used to create a bot for a resource gathering scenario (Section 4) followed by a more complex tankbattle scenario in Section 5. We will also present some preliminary results on how to deal with imperfect information, Fog of War (Section 6). The methodology has been presented in our previous papers [11, 12]. This article summarises the previous work and extends it by adding new experiments and new results. Last in this article, we have a discussion and line out some directions for future work.

2. A Methodology for Multiagent Potential Fields

When constructing a multiagent potential field-based system for controlling agents in a certain domain, there are a number of issues that we must take into consideration. It is, for example, important that each interesting object in the game world generates some type of field, and we must decide which objects can use static fields to decrease computation time.

To structure this, we identify six phases in the design of an MAPF-based solution:

(1) the identification of objects;

(2) the identification of the driving forces (i.e., the fields) of the game;

(3) the process of assigning charges to the objects;

(4) the granularity of time and space in the environment;

(5) the agents of the system;

(6) the architecture of the MAS.

In *the first phase*, we may ask us the following questions. What are the *static objects* of the environment? That is, what objects keep their attributes throughout the lifetime of the scenario? What are the *dynamic objects* of the environment? Here we may identify a number of different ways that objects may change. They may move around, if the environment has a notion of physical space. They may change their attractive (or repulsive) impact on the agents. What is the *modifiability* of the objects? Some objects may be consumed, created, or changed by the agents.

In *the second phase*, we identify the driving forces of the game at a rather abstract level, for example, to avoid obstacles, or to base the movements on what the opponent does. This leads us to a number of fields. The main reason to enable multiple fields is that it is very easy to isolate certain aspects of the computation of the potentials if we are able to filter out a certain aspect of the overall potential, for example, the repulsive forces generated by the terrain in a physical environment. We may also dynamically weight fields separately, for example, in order to decrease the importance of the navigation field when a robot stands still in a surveillance mission (and only moves its camera). We may also have *strategic fields* telling the agents in what direction their next goal is, or *tactical fields* coordinating the movements with those of the teammate agents.

The third phase includes placing the objects in the different fields. Static objects should typically be in the *field of navigation*. The potentials of such a field are precalculated in order to save precious run time CPU resources.

In *the fourth phase*, we have to decide the resolution of space and time. Resolution of space means how detailed the navigation in the game world should be. Should for example the agents be able to move to every single point in the world, or should the game world be divided into a grid with tiles of for example 4×4 points in the world? Resolution of time means how often the potential fields should be updated. If the agents are able to move around in the environment, both these measures have an impact on the lookahead. The space resolution obviously, since it decides what points in space that we are able to access, and the time in that it determines how far we may get in one time frame (before it is time to make the next decision about what to do).

The fifth phase is to decide what objects to agentify and set the repertoire of those agents: what actions are we going to evaluate in the lookahead? As an example, if the agent is omnidirectional in its movements, we may not want to evaluate all possible points that the agent may move to, but rather try to filter out the most promising ones by using some heuristic, or use some representable sample.

In *the sixth step*, we design the architecture of the MAS. Here we take the unit agents identified in the fifth phase, give them roles, and add the supplementary agents (possibly) needed for coordination, and special missions (not covered by the unit agents themselves).

3. ORTS

Open real-time strategy (ORTS) [13] is a real-time strategy game engine developed as a tool for researchers within artificial intelligence (AI) in general and game AI in particular. ORTS uses a client-server architecture with a game server and players connected as clients. Each timeframe clients receives a data structure from the server containing the current game state. Clients can then call commands that activate and control their units. Commands can be like "move unit A to (x, y) or attack opponent unit X with unit A." The game server executes the client commands in random order.

Users can define different types of games in scripts where units, structures, and their interactions are described. All

types of games from resource gathering to full real-time strategy (RTS) games are supported.

We will begin by looking at a one-player resource gathering scenario game called *Collaborative Pathfinding*, which was part of the 2007 and 2008 ORTS competitions [13]. In this game, the player has 20 worker units. The goal is to use the workers to mine resources from nearby mineral patches and return them to a base. A worker must be adjacent to a mineral object to mine, and to a base to return resources. As many resources as possible will be collected within 10 minutes.

This is followed by looking at the two-player games, *Tankbattle*, which was part of the 2007 and 2008 ORTS competitions [13] as well.

In *Tankbattle,* each player has 50 tanks and five bases. The goal is to destroy the bases of the opponent. Tanks are heavy units with long fire range and devastating firepower but a long cool-down period, that is, the time after an attack before the unit is ready to attack again. Bases can take a lot of damage before they are destroyed, but they have no defence mechanism of their own so it may be important to defend our own bases with tanks. The map in a tankbattle game has randomly generated terrain with passable lowland and impassable cliffs.

Both games contain a number of neutral units (sheep). These are small indestructible units moving randomly around the map. The purpose of sheep is to make pathfinding and collision detection more complex.

4. Multiagent Potential Fields in ORTS

First we will describe a bot playing the Collaborative Pathfinding game based on MAPF following the proposed methodology. Collaborative Pathfinding is a 1-player game where the player has one control center and 20 worker units. The aim is to move workers to mineral patches, mine up to 10 resources (the maximum load a worker can carry), then return to a friendly control center to drop them off.

4.1. Identifying Objects. We identify the following objects in our application: *Cliffs, Sheep, Base stations,* and *workers.*

4.2. Identifying Fields. We identified five tasks in ORTS: avoid colliding with the terrain, avoid getting stuck at other moving objects, avoid colliding with the bases, move to the bases to leave resources, and move to the mineral patches to get new resources. This leads us to three major types of potential fields: a *field of navigation*, a *strategic field,* and a *tactical field*.

The field of navigation is a field generated by repelling static terrain. This is because we would like the agents to avoid getting too close to objects where they may get stuck, but instead smoothly pass around them.

The strategic field is a dynamic attracting field. It makes agents go towards the mineral patches to mine, and return to the base to drop off resources.

Own workers, bases, and sheep generate small repelling fields. The purpose of these fields is the same as for obstacle avoidance; we would like our agents to avoid colliding with each other and bases as well as avoiding the sheep. This task is managed by the tactical field.

4.3. Assigning Charges. Each worker, base, sheep, and cliffs has a set of charges which generates a potential field around the object. These fields are weighted and summed together to form a total potential field that is used by our agents for navigation.

Cliffs, for example, impassable terrain, generate a repelling field for obstacle avoidance. The field is constructed by copying pregenerated matrixes of potentials into the field of navigation when a new game is started. The potential all cliffs generate in a point (x, y) is calculated as the lowest potential a cliff generates in that point. The potential $p_{\text{cliff}}(d)$ in a point at distance d from the closest impassable terrain tile is calculated as:

$$p_{\text{cliff}}(d) = \begin{cases} \dfrac{-80}{(d/8)^2} & \text{if } d > 0, \\ -80 & \text{if } d = 0. \end{cases} \tag{1}$$

Own worker units generate repelling fields for obstacle avoidance. The potential $p_{\text{worker}}(d)$ at distance d from the center of another worker is calculated as

$$p_{\text{worker}}(d) = 1.429 \cdot \begin{cases} -20 & \text{if } d \leq 6, \\ 16 - 2 \cdot d & \text{if } d \in]6, 8]. \end{cases} \tag{2}$$

Sheep. Sheep generate a small repelling field for obstacle avoidance. The potential $p_{\text{sheep}}(d)$ at distance d from the center of a sheep is calculated as

$$p_{\text{sheep}}(d) = 0.125 \cdot \begin{cases} -20 & \text{if } d \leq 8, \\ 2 \cdot d - 25 & \text{if } d \in]8, 12.5]. \end{cases} \tag{3}$$

Own bases. The own bases generate two different fields depending on the current state of a worker. The base generates an attractive field if the worker needs to move to the base and drop off its resources. Once it has arrived at the base, all the resources are dropped. The potential $p_{\text{attractive}}(d)$ at distance d from the center of the base is calculated as

$$p_{\text{attractive}}(d) = \begin{cases} 240 - d \cdot 0.32 & \text{if } d \leq 750, \\ 0 & \text{if } d > 750. \end{cases} \tag{4}$$

In all other states of the worker, the own base generates a repelling field for obstacle avoidance. Below is the function for calculating the potential $p_{\text{own }B}(d)$ at distance d from the center of the base. Note that this is, of course, the view of the worker. The base will effect some of the workers with the attracting field while at the same time effect the rest with a repelling field. If a point is inside the quadratic area the base occupies, the potential in those points is always 10000 (potential used for impassable points):

$$p_{\text{own }B}(d) = 0.125 \cdot \begin{cases} 6 \cdot d - 258 & \text{if } d \leq 43, \\ 0 & \text{if } d > 43. \end{cases} \tag{5}$$

Minerals, similar to own bases, generate attractive fields for all workers that do not carry maximum loads and a

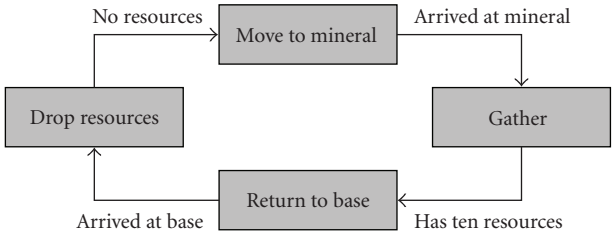


Figure 1: The finite state machine used by the workers in a resource gathering scenario.

repelling field for obstacle avoidance when they do. The potential of the attractive field is the same as the attractive field around the own base in (4).

In the case when minerals generate a repelling field, the potential $p_{\text{mineral}}(d)$ at distance d from the center of a mineral is calculated as

$$p_{\text{mineral}}(d) = 1.429 \cdot \begin{cases} -20 & \text{if } d \leq 8, \\ 20 - 2 \cdot d & \text{if } d \in]8, 10]. \end{cases} \quad (6)$$

4.4. The Granularity of the System. Since the application is rather simple, we use full resolution of both the map and the time frames without any problems.

4.5. The Agents. The main units of our system are the workers. They use a simple finite state machine (FSM) illustrated in Figure 1 to decide what state they are in (and thus what fields to activate). No central control or explicit coordination is needed, since the coordination is emerging through the use of the charges.

4.6. The Multiagent System Architecture. In addition to the worker agents, we have one additional agent that is the interface between the workers and the game server. It receives server information about the positions of all objects and workers which it distributes to the worker agents. They then decide what to do, and submit their proposed actions to the interface agent which in turn sends them through to the ORTS server.

4.7. Experiments, Resource Gathering. Table 1 shows the result from the Collaborative Pathfinding game in 2008 years' ORTS tournament. It shows that an MAPF-based bot can compete with A*-based solutions in a resource gathering scenario. There are however some uncertainties in these results. Our bot has disconnected from the server (i.e., crashed) in 30 games. The reason for this is not yet clear and must be investigated in more detail. Another issue is that Uofa has used the same bot that they used in the 2007 years' tournament, and the bot had a lower score this year. The reason, according to the authors, was "probably caused by a pathfinding bug we introduced" [14]. Still we believe that with some more tuning and bug fixing our bot can probably match the best bots in this scenario.

Table 1: Experiment results from the Collaborative Pathfinding game in 2008 years' tournament.

Team	Matches	Avg. Resources	Disconnected
BTH	250	5630.72	30
Uofa	250	4839.6	0

5. MAPF in ORTS, Tankbattle

In the 2-player Tankbattle game, each player has a number of tanks and bases, and the goal is to destroy the opponent bases. In [11] we describe the implementation of an ORTS bot playing Tankbattle based on MAPF following the proposed methodology. This bot was further improved in [12] where a number of weaknesses of the original bot were addressed. We will now, just as in the case of the Collaborative pathfinding scenario, present the six steps of the used methodology. However, there are details in the implementation of several of these steps that we have improved and shown the effect of in experiments. We will therefore, to improve the flow of the presentation, not present all of them in chronologic order. Instead we start by presenting the ones that we have kept untouched through the series of experiments.

5.1. Identifying Objects. We identify the following objects in our application: Cliffs, Sheep, and own (and opponent) tanks and base stations.

5.2. Identifying Fields. We identified four tasks in ORTS: *Avoid colliding with the terrain, Avoid getting stuck at other moving objects, Hunt down the enemy's forces,* and *Defend the bases*. In the resource gathering scenario we used the two major types: *field of navigation* and *strategic field*. Here we add a new major type of potential field: the *defensive field*.

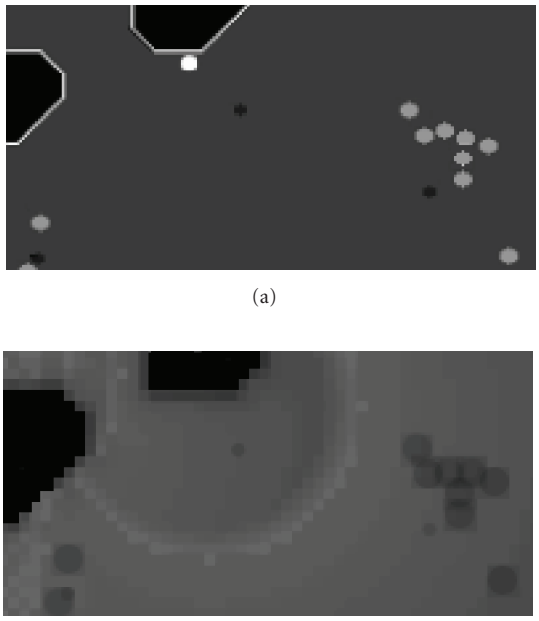
The field of navigation is, as in the previous example of Collaborative pathfinding, a field generated by repelling static terrain for obstacle avoidance.

The strategic field is an attracting field. It makes units go towards the opponents and place themselves on appropriate distances where they can fight the enemies.

The defensive field is a repelling field. The purpose is to make own agents retreat from enemy tanks when they are in cooldown phase. After an agent has attacked an enemy unit or base, it has a cooldown period when it cannot attack and it is therefore a good idea to stay outside enemy fire range while being in this phase. The defensive field is an improvement to deal with a weakness found in the original bot [11].

Own units, own bases, and sheep generate small repelling fields. The purpose is the same as for obstacle avoidance; we would like our agents to avoid colliding with each other or bases as well as avoiding the sheep. This is managed by the tactical field.

5.3. Assigning Charges. The upper picture in Figure 2 shows part of the map during a tankbattle game. The screenshots are from the 2D GUI available in the ORTS server. It shows our agents (light-grey circles) moving in to attack an



(a)

(b)

Figure 2: Part of the map during a tankbattle game. The upper picture shows our agents (light-grey circles), an opponent unit (white circle), and three sheep (small dark-grey circles). The lower picture shows the total potential field for the same area. Light areas have high potential and dark areas have low potential.

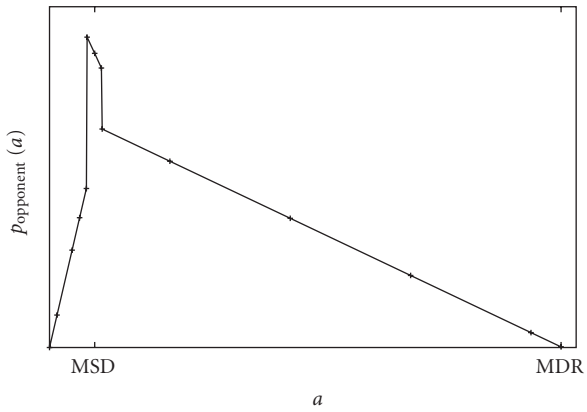


Figure 3: The potential $p_{\text{opponent}}(a)$ generated by opponent units as a function of the distance a.

opponent unit (white circle). The area also has some cliffs (black areas) and three sheep (small dark-grey circles). The lower picture shows the total potential field in the same area. Dark areas have low potential and light areas have high potential. The light ring around the opponent unit, located at maximum shooting distance of our tanks, is the distance from which our agents prefer to attack opponent units. The picture also shows the small repelling fields generated by our own agents and the sheep.

Cliffs. Cliffs generate the same field as in the resource gathering scenario, see Section 4.3.

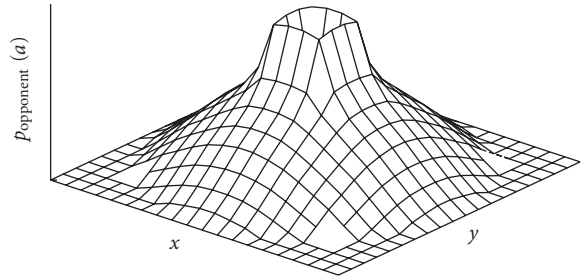


Figure 4: The potential $p_{\text{opponent}}(a)$ generated by the opponent that is in the middle.

The Opponent Units and Bases. All opponent units and bases generate symmetric surrounding fields where the highest potential is in a ring around the object with a radius of *maximum shooting distance* (MSD). MDR refers to the *Maximum Detection Range*, the distance from which an agent starts to detect the opponent unit. The reason why the location of the enemy unit is not the final goal is that we would like our units to surround the enemy units by attacking from the largest possible distance. The potential all opponent units generate in a certain point is then equal to the highest potential *any* opponent unit generates in that point, and not the sum of the potentials that all opponent units generate. If we were to sum the potentials, the highest potential and most attractive destination would be in the center of the opponent unit cluster. This was the case in the first version of our bot and was identified as one of its major weaknesses [11]. The potentials $p_{\text{opp}U}(d)$ and $p_{\text{opp}B}(d)$ at distance d from the center of an agent and with $D = \text{MSD}$ and $R = \text{MDR}$ are calculated as

$$p_{\text{opp}U}(d) = 0.125 \cdot \begin{cases} 240/d(D-2) & \text{if } d \in [0, D-2[, \\ 240 & \text{if } d \in [D-2, D], \\ 240 - 0.24(d-D) & \text{if } d \in]D, R], \end{cases}$$

$$p_{\text{opp}B}(d) = 0.125 \cdot \begin{cases} 360/d(D-2) & \text{if } d \in [0, D-2[, \\ 360 & \text{if } d \in [D-2, D], \\ 360 - 0.32(d-D) & \text{if } d \in]D, R]. \end{cases} \tag{7}$$

$I = [a, b[$ denote the half-open interval, where $a \in I$, but $b \notin I$.

Own units generate repelling fields for obstacle avoidance. The potential $p_{\text{own}U}(d)$ at distance d from the center of a unit is calculated as:

$$p_{\text{own}U}(d) = 0.125 \cdot \begin{cases} -20 & \text{if } d \le 14 \\ 32 - 2 \cdot d & \text{if } d \in]14, 16] \end{cases} \tag{8}$$

Own bases generate repelling fields similar to the fields around the own bases described in Section 4.3.

Sheep generate the same weak repelling fields as in the Collaborative pathfinding scenario, see Section 4.3.

5.4. The Multiagent Architecture. In addition to the interface agent dealing with the server (which is more or less the

Table 2: Experiment results from the original bot.

Team	Wins ratio	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	0%	(0/100)	0.01	0.00	−46.99
WarsawB	0%	(0/100)	1.05	0.01	−42.56
UBC	24%	(24/100)	4.66	0.92	−17.41
Uofa.06	32%	(32/100)	4.20	1.45	−16.34
Average	14%	(14/100)	2.48	0.60	−30.83

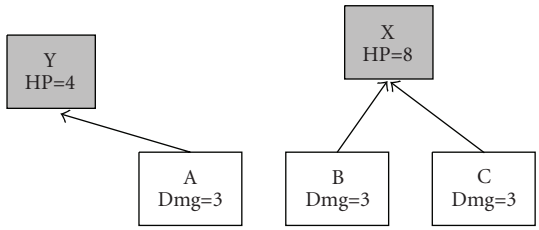


Figure 5: Attacking most damaged unit within firerange.

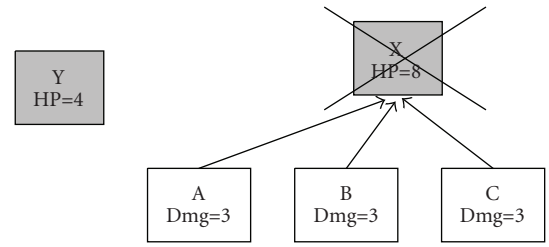


Figure 6: Optimise attacks to destroy as many units as possible.

same as in the collaborative pathfinding scenario), we use a coordinator agent to globally coordinate the attacks on opponent units to maximise the number of opponent units destroyed. The difference between using the coordinator agent compared to attacking the most damaged unit within fire range is best illustrated with an example.

In Figure 5, the own units A, B, and C does 3 damage to opponent units. They can attack opponent unit X (can take 8 more damage before it is destroyed) and unit Y (can take 4 more damage before it is destroyed). Only unit A can attack enemy unit Y. The most common approach in the ORTS tournament [13] was to attack the most damaged enemy unit within firerange. In the example both enemy unit X and Y would be attacked, but both would survive to answer the attacks.

With the coordinator agent attacks would be spread out as in Figure 6. In this case enemy unit X would be destroyed and only unit Y can answer the attacks.

5.5. The Granularity of the System. Each unit (own or enemy), base, sheep, and cliffs has a set of charges which generates a potential field around the object. These fields are weighted and summed together to form a total potential field that is used by our agents for navigation.

In [11] we used pregenerated fields that were simply added to the total potential field at runtime. To reduce memory and CPU resources needed, the game world was split into tiles where each tile was 8×8 points in the game world. This proved not to be detailed enough and our agents often got stuck in terrain and other game objects. The results as shown in Table 2 are not very impressive and our bot only won 14% of the played games.

Some notes on how the results are presented:

(i) *Avg units.* This is the average number of units (tanks) our bot had left after a game is finished.

(ii) *Avg bases.* This is the average number of bases our bot had left after a game is finished.

(iii) *Avg score.* This is the average score for our bot after a game is finished. The score is calculated as

$$\begin{aligned} \text{score} = {} & 5(\text{ownBasesLeft} - \text{oppBasesLeft}) \\ & + \text{ownUnitsLeft} - \text{oppUnitsLeft}. \end{aligned} \tag{9}$$

In [12] we proposed a solution to this problem. Instead of dividing the game world into tiles, the resolution of the potential fields was set to 1×1 points. This allows navigation at the most detailed level. To make this computationally feasible, we calculate the potentials at runtime, but only for those points that are near own units that are candidates to move to in the next time frame. In total, we calculate nine potentials per unit, eight directions, and the potential of staying in the position it is. The results, as shown in Table 3, show a slight increase in the number of games won and a large improvement in the game score.

5.6. Adding an Additional Field. Defensive Field. After a unit has fired its weapon, the unit has a cooldown period when it cannot attack. In the original bot our agents were, as long as there were enemies within maximum shooting distance (MSD), stationary until they were ready to fire again. The cooldown period can instead be used for something more useful and in [12] we proposed the use of a defensive field. This field makes the units retreat when they cannot attack and advance when they are ready to attack once again. With this enhancement, our agents always aim to be at MSD of the closest opponent unit or base and surround the opponent unit cluster at MSD. The potential $p_{\text{defensive}}(d)$ at distance d from the center of an agent is calculated using the formula in

$$p_{\text{defensive}}(d) = \begin{cases} w_2 \cdot (-800 + 6.4 \cdot d) & \text{if } d \leq 125, \\ 0 & \text{if } d > 125. \end{cases} \tag{10}$$

Table 3: Experiment results from increased granularity.

Team	Wins ratio	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	9%	(9/100)	1.18	0.57	−32.89
WarsawB	0%	(0/100)	3.03	0.12	−36.71
UBC	24%	(24/100)	16.11	0.94	0.46
Uofa.06	42%	(42/100)	10.86	2.74	0.30
Average	18.75%	(18.75/100)	7.80	1.09	−17.21

Table 4: Experiment results from defensive field.

Team	Wins ratio	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	64%	(64/100)	22.95	3.13	28.28
WarsawB	48%	(48/100)	18.32	1.98	15.31
UBC	57%	(57/100)	30.48	1.71	29.90
Uofa.06	88%	(88/100)	29.69	4.00	40.49
Average	64.25%	(64.25/100)	25.36	2.71	28.50

The use of a defensive field is a great performance improvement of the bot, and it now wins over 64% of the games against the four opponent teams (Table 4).

5.7. Local Optima. To get stuck in local optima is a problem that is well known and that has to be dealt with when using PF. Local optima are positions in the potential field that have higher potential than all their neighbouring positions. An agent positioned in a local optimum may therefore get stuck even if the position is not the final destination for the agent. In the first version of our bot, agents that had been idle for some time moved in random directions for some frames. This is not a very reliable solution to the problem since there are no guarantees that the agents will move out of, or will not directly return to, the local optima.

Thurau et al. [15] describe a solution to the local optima problem called *avoid-past potential field forces*. In this solution, each agent generates a trail of negative potentials on previous visited positions, similar to a pheromone trail used by ants. The trail pushes the agent forward if it reaches a local optimum. We have introduced a trail that adds a negative potential to the last 20 positions of each agent. Note that an agent is not affected by the trails of other own agents. The negative potential used for the trail is set to −0.5.

The use of pheromone trails further boosts the result and our bot now wins 76.5% of the games (see Table 5).

5.8. Using Maximum Potentials. In the original bot, all potential fields generated by opponent units were weighted and summed to form the total potential field which is used for navigation by our agents. The effect of summing the potential fields generated by opponent units is that the highest potentials are generated from the centres of the opponent unit clusters. This makes our agents attack the centres of the enemy forces instead of keeping the MSD to the *closest* enemy. The proposed solution to this issue is that, instead of summing the potentials generated by opponent units and bases, we add the highest potential any opponent unit or base generates. The effect of this is that our agents engage the closest enemy unit at maximum shooting distance instead of trying to keep the MSD to the centre of the opponent unit cluster. The results from the experiments are presented in Table 6.

5.9. A Final Note on the Performance. Our results were further validated in the 2008 ORTS tournament, where our PF-based bots won the three competitions that we participated in (Collaborative Pathfinding, Tankbattle, and Complete RTS). In the Tankbattle competition, we won all 100 games against NUS, the winner of last year, and only lost four of 100 games to Lidia (see Table 7 [14]).

6. Fog of War

To deal with FoW, the bot needs to solve the following issues: remember locations of enemy bases, explore unknown terrain to find enemy bases and units, and handle dynamic terrain due to exploration. We must also take into consideration the increase in computational resources needed when designing solutions to these issues. To enable FoW for only one client, we made a minor change in the ORTS server. We added an extra condition to an IF statement that always enabled Fog of War for client 0. Due to this, our client is always client 0 in the experiments (of course, it does not matter from the game point of view if the bots play as client 0 or client 1). The changes we made to deal with these issues come below.

6.1. Remember Locations of the Enemies. In ORTS, a data structure with the current game world state is sent, each frame from the server to the connected clients. If Fog of War is enabled, the location of an enemy base is only included in the data structure if an own unit is within the visibility range of the base. It means that an enemy base if has been spotted by an own unit and that unit is destroyed, the location of the base is no longer sent in the data structure. Therefore our bot

Table 5: Experiment results from avoid-past potential field forces.

Team	Wins ratio	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	73%	(73/100)	23.12	3.26	32.06
WarsawB	71%	(71/100)	23.81	2.11	27.91
UBC	69%	(69/100)	30.71	1.72	31.59
Uofa.06	93%	(93/100)	30.81	4.13	46.97
Average	76.5%	(76.5/100)	27.11	2.81	34.63

Table 6: Experiment results from using maximum potential, instead of summing the potentials.

Team	Win %	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	100%	(100/100)	28.05	3.62	46.14
WarsawB	99%	(99/100)	31.82	3.21	47.59
UBC	98%	(98/100)	33.19	2.84	46.46
Uofa.06	100%	(100/100)	33.19	4.22	54.26
Average	99.25%	(99.25/100)	31.56	3.47	48.61

Table 7: Results from the ORTS Tankbattle 2008 competition.

Team	Total win %	Blekinge	Lidia	NUS
Blekinge	98	—	96	100
Lidia	43	4	—	82
NUS	9	0	18	—

has a dedicated global map agent to which all detected objects are reported. This agent always remembers the location of previously spotted enemy bases until a base is destroyed, and distributes the positions of detected enemy tanks to all the own units.

The global map agent also takes care of the map sharing concerning the opponent tank units. However, it only shares momentary information about opponent tanks that are within the detection range of at least one own unit. If all units that see a certain opponent tank are destroyed, the position of that tank is no longer distributed by the global map agent and that opponent disappears from our map.

6.2. Dynamic Knowledge about the Terrain. If the game world is completely known, the knowledge about the terrain is static throughout the game. In the original bot, we created a static potential field for the terrain at the beginning of each new game. With Fog of War, the terrain is partly unknown and must be explored. Therefore our bot must be able to update its knowledge about the terrain.

Once the distance to the closest impassable terrain has been found, the potential is calculated as

$$p_{\text{terrain}}(d) = \begin{cases} -10000 & \text{if } d \leq 1, \\ \dfrac{-5}{(d/8)^2} & \text{if } d \in]1, 50], \\ 0 & \text{if } d > 50. \end{cases} \tag{11}$$

6.3. Exploration. Since the game world is partially unknown, our units have to explore the unknown terrain to locate the hidden enemy bases. The solution we propose is to assign an attractive field to each unexplored game tile. This works well in theory as well as in practice if we are being careful about the computation resources spent on it.

The potential p_{unknown} generated in a point (x, y) is calculated as follows.

(1) Divide the terrain tile map into blocks of 4×4 terrain tiles.

(2) For each block, check every terrain tile in the block. If the terrain is unknown in ten or more of the (at most 16) checked tiles the whole block is considered unknown.

(3) For each block that needs to be explored, calculate the Manhattan Distance md from the center of the own unit to the center of the block.

(4) Calculate the potential p_{unknown} each block generates using (12) below.

(5) The total potential in (x, y) is the sum of the potentials each block generates in (x, y):

$$p_{\text{unknown}}(\text{md}) = \begin{cases} \left(0.25 - \dfrac{\text{md}}{8000}\right) & \text{if md} \leq 2000, \\ 0 & \text{if md} > 2000. \end{cases} \tag{12}$$

6.4. Experiments, FoW Bot. In this experiment set we have used the same setup as in the Tankbattle except that now our bot has FoW enabled, that is, it does not get information about objects, terrain, and so forth that is further away than 160 points from all of our units. At the same time, the opponents have complete visibility of the game world. The results of the experiments are presented in Table 8. They show that our bot still wins 98.5% of the games against the opponents, which is just a minor decrease compared to having complete visibility.

It is also important to take into consideration the changes in the needs for computational resources when FoW is

Table 8: Experiment results when FoW is enabled for our bot.

Team	Wins ratio	Wins/games	Avg. units	Avg. bases	Avg. score
NUS	100%	(100/100)	29.74	3.62	46.94
WarsawB	98%	(98/100)	32.35	3.19	46.70
UBC	96%	(96/100)	33.82	3.03	47.67
Uofa.06	100%	(100/100)	34.81	4.27	54.90
Average	98.5%	(98.5/100)	32.68	3.53	49.05

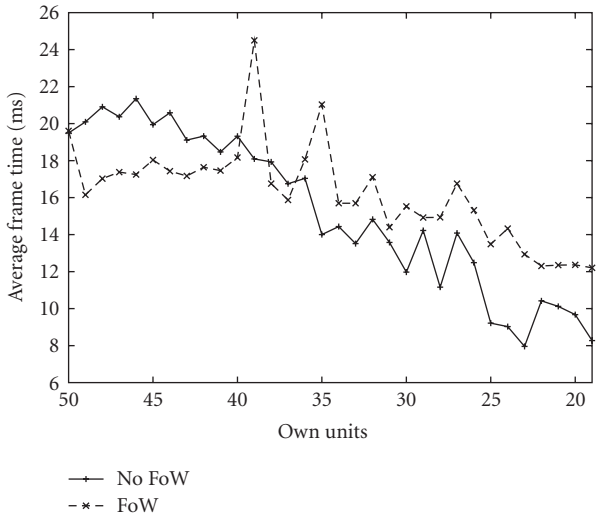


Figure 7: The average frame time used for bots with perfect and imperfect information about the game world.

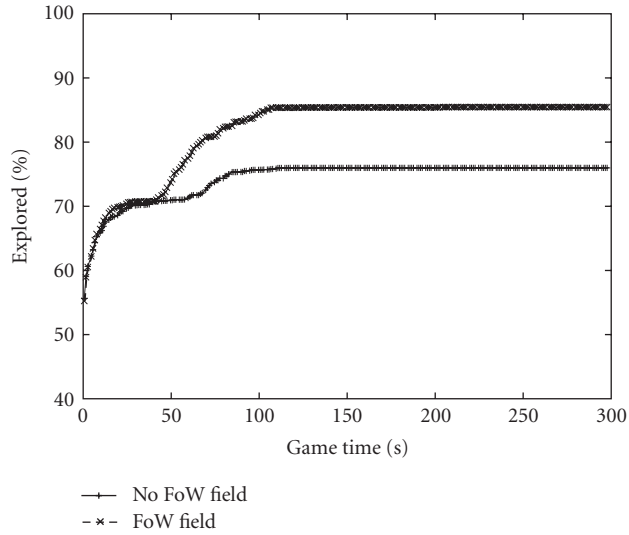


Figure 8: The average explored area given the current game time for a bot using the field of exploration, compared to one that does not.

enabled, since we need to deal with dynamic terrain and exploration field. To show this we have run 100 games without FoW against team NUS and 100 games with FoW enabled. The same seeds are used for both. For each game we measured the average time in milliseconds that the bots used in each game frame and the number of own units left. Figure 7 shows the average frame time for both bots in relation to number of own units left. It shows that the FoW-enabled bot used *less* CPU resources in the beginning of a game, which is probably because some opponent units and bases are hidden in unexplored areas and less potential field-based on opponent units have to be generated. Later in the game, the FoW bot requires more CPU resources probably due to the exploration and the dynamic terrain fields.

In the next set of experiments we show the performance of the exploration field. We ran 20 different games in this experiment, each in which the opponent faced both a bot with the field of exploration enabled, and one where this field was disabled (the rest of the parameters, seeds, etc. were kept identical). Figure 8 shows the performance of the exploration field. It shows how much area that has been explored given the time of the game. The standard deviation increases with the time since only a few of the games last longer than three minutes.

In Table 9, we see that the use of the field of exploration (as implemented here) does not improve the results dramatically. However, the differences are not statistically significant.

Table 9: Performance of the bot with and without field of exploration in 20 matches against NUS.

Version	Won	Lost	Avg. units	Avg. bases
With FoE	20	0	28.65	3.7
Without FoE	19	1	27.40	3.8

7. Discussion

We have shown that the bot can easily be modified to handle changes in the environment, in this case a number of details concerning the agents, the granularity, the fields, and also FoW. The results show that FoW initially decreases the need for processing power and in the end, it had a very small impact on the performance of the bot in the matches. However, this has to be investigated further. In Figure 8, we see that using the field of exploration in general gives a higher degree of explored area in the game, but the fact that the average area is not monotonically increasing as the games go on may seem harder to explain. One plausible explanation is that the games where our units do not get stuck in the terrain will be won faster as well as having more units available to explore the surroundings. When these games end, they do not contribute to the average and the average difference in explored areas will decrease. Does the field of exploration contribute to the performance? Is it at all important to be

able to explore the map? Our results (see Table 9) indicate that—it in this case—may not be that important. However, the question is complex. Our experiments were carried out with an opponent bot that had perfect information and thus was able to find our units. The results may have been different if also the opponent lacked perfect information.

It is our belief that MAPF-based bots in RTS games have great potential even though the scenarios used in the experiments are, from an AI perspective, quite simple RTS scenarios. In most modern commercial RTS games, the AI (and human player) has to deal with base building, economics, technological development, and resource gathering. However, we cannot think of any better testbed for new and innovative RTS games AI research than to test it in competitions like ORTS.

8. Conclusions and Future Work

In Section 4 we introduced a methodology for creating MAPF-based bots in an RTS environment. We showed how to deal with a gathering resources scenario in an MAPF-based bot. Our bot won this game in the 2008 years' ORTS competition, but would have ended up somewhere in the middle in 2007 years' tournament. The bot had some problems with crashes, and more work can be done here to further boost the result.

This was followed by Section 5 where we showed how to design an MAPF-based for playing a tankbattle game. The performance of the first version of our bot was tested in the 2007 years' ORTS competition organized by the University of Alberta. The results, although not very impressive, showed that the use of MAPF-based bots had potential. A number of weaknesses of the first version were identified, solutions to these issues were proposed and new experiments showed that the bot won over 99% of the games against four of the top teams from the tournament. This version of the bot won the 2008 years' tournament with an almost perfect score of 98% wins.

Some initial work has been done in this direction. Our bot quite easily won the full RTS scenario in the 2008 years' ORTS tournament, but more has to be done here. The full RTS scenario in ORTS, even though handling most parts of a modern RTS game, is still quite simple. We will develop this in the future to handle a larger variety of RTS game scenarios.

Another potential idea is to use the fact that our solution, in many ways, is highly configurable even in runtime. By adjusting weights of fields, the speed of the units, and so forth in real time, the performance can be more or less changed as the game goes on. This can be used to tune the performance to the level of the opponent to create games that are more enjoyable to play. One of our next projects will focus on this aspect of MAPF-based bots for RTS games.

Acknowledgments

We would like to thank Blekinge Institute of Technology for supporting our research, the anonymous reviewers, and the organisers of ORTS for providing an interesting application.

References

[1] A. Nareyek, "AI in computer games," *Queue*, vol. 1, no. 10, pp. 58–65, 2004.

[2] 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.

[3] R. C. Arkin, "Motor schema based navigation for a mobile robot: an approach to programming by behavior," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA '87)*, vol. 4, pp. 264–271, Raleigh, NC, USA, March 1987.

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

[5] M. Massari, G. Giardini, and F. Bernelli-Zazzera, "Autonomous navigation system for planetary exploration rover based on artificial potential fields," in *Proceedings of the 6th Conference on Dynamics and Control of Systems and Structures in Space (DCSSS '04)*, Riomaggiore, Italy, July 2004.

[6] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[7] A. Howard, M. Matarić, and G. Sukhatme, "Mobile sensor network deployment using potential fields: a distributed, scalable solution to the area coverage problem," in *Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems (DARS '02)*, Fukuoka, Japan, June 2002.

[8] S. J. Johansson and A. Saffiotti, "An electric field approach to autonomous robot control," in *Robot Soccer World Cup V (RoboCup '01)*, Springer, London, UK, 2002.

[9] T. Röfer, R. Brunn, I. Dahm, et al., GermanTeam 2004: the german national Robocup team.

[10] C. Thurau, C. Bauckhage, and G. Sagerer, "Learning human-like movement behavior for computer games," in *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB '04)*, Los Angeles, Calif, USA, July 2004.

[11] J. Hagelbäck and S. J. Johansson, "Using multiagent potential fields in real-time strategy games," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '08)*, L. Padgham and D. Parkesm, Eds., vol. 2, pp. 631–638, Estoril, Portugal, May 2008.

[12] J. Hagelbäck and S. J. Johansson, "The rise of potential fields in real time strategy bots," in *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE '08)*, Stanford, Calif, USA, October 2008.

[13] M. Buro, "ORTS—A Free Software RTS Game Engine," July 2007, http://www.cs.ualberta.ca/~mburo/orts/.

[14] M. Buro, "ORTS RTS game AI competition," August 2008, http://www.cs.ualberta.ca/~mburo/orts/AIIDE08/.

[15] C. Thurau, C. Bauckhage, and G. Sagerer, "Imitation learning at all levels of game-ai," in *Proceedings of the 5th International Conference on Computer Games, Artificial Intelligence, Design and Education (CGAIDE '04)*, pp. 402–408, University of Wolverhampton, Reading, UK, November 2004.