Using Multi-agent Potential Fields in Real-time Strategy Games

Johan Hagelbäck
Department of Software and Systems Engineering
Blekinge Institute of Technology
Box 520, SE-372 25, Ronneby, Sweden
jhg@bth.se

Stefan J. Johansson
Department of Software and Systems Engineering
Blekinge Institute of Technology
Box 520, SE-372 25, Ronneby, Sweden
sja@bth.se

ABSTRACT

Bots for Real Time Strategy (RTS) games provide a rich challenge to implement. A bot controls a number of units that may have to navigate in a partially unknown environment, while at the same time search for enemies and coordinate attacks to fight them down. Potential fields is 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 Multi-agent Potential Field based bot architecture that is evaluated in a real time strategy game setting and compare it, both in terms of performance, and in terms of softer attributes such as configurability with other state-of-the-art solutions. Although our solution did not reach the performance standards of traditional RTS bots in the test, we see great unexploited benefits in using multi-agent potential field based solutions in RTS games.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence, Distributed Artificial Intelligence - *Multi-agent systems* and I.2.1 [**Computing Methodologies**]: Artificial Intelligence, Applications and Expert Systems - *Games*

General Terms

Algorithms, Design, Performance

Keywords

Artificial Potential Fields, RTS Games, ORTS, Multi-agent Bot

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.

Cite as: Using Multi-agent Potential Fields in Real-time Strategy Games, Johan Hagelbäck and Stefan J. Johansson, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes,MüllerandParsons(eds.),May,12-16.,2008,Estoril,Portugal, pp.631-638.

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 [8]. Later on Arkin [1] updated the knowledge by creating another technique using superposition of spatial vector fields in order to generate behaviours in his so called motor schema concept.

Many studies concerning potential fields are related to spatial navigation and obstacle avoidance, see e.g. [3, 9, 12]. 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 [2]. However most of the premises of these approaches are only based on repulsive potential fields of the obstacles and an attractive potential in some goal for the robot [17].

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 [5], and potential fields have been used in robot soccer [7, 14]. Thurau et al. [15] has developed a game bot which learns reactive behaviours (or potential fields) for actions in the First-Person Shooter (FPS) game Quake II through imitation.

In some respect, videogames are perfect test platforms for multi-agent systems. The environment may be competitive (or even hostile) as in the case of a FPS game. The NPCs (e.g. the units of the opponent army in a war strategy game) are supposed to act rationally and autonomously, and the units act in an environment which enables explicit communication and collaboration in order to be able to solve certain tasks.

Previous work on describing how intelligent agent technology has been used in videogames include the extensive survey of Niederberger and Gross [13] and early work by van Lent et al. [18]. Multi-agent systems has been used in board games by Kraus and Lehmann who addressed the use of MAS in Diplomacy [10] and Johansson who proposed a general MAS architecture for board games [6].

The main research question of this paper is: *Is Multi-agent Potential Fields (*MAPF*) an appropriate approach to implement highly configurable bots for* RTS *games?* This breaks down to:

1. How does MAPF perform compared to traditional solutions?
2. To what degree is MAPF an approach that is configurable with respect to variations in the domain?

We will use a proof of concept as our main methodology where we compare an implementation of MAPF playing ORTS with other approaches to the game. The comparisons are based both on practical performance in the yearly ORTS tournament, and some theoretical comparisons based on the descriptions of the other solutions.

First we describe the methodology that we propose to follow for the design of a MAPF bot. In Section 3 we describe the test environment. The creation of our MAPF player follows the proposed methodology and we report on that in Section 4. The experiments and their results are described in Section 5. We finish off by discussing, drawing some conclusions and outlining future work in Sections 6–7.

2. A METHODOLOGY FOR MULTI-AGENT POTENTIAL FIELDS

When constructing a multi-agent system of potential field controlled agents in a certain domain, there are a number of issues that have to be dealt with. To structure this, we identify six phases in the design of a MAPF-based solution:

1. The identification of objects,
2. The identification of the driving forces (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, and
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 remain 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 are 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, e.g. 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, e.g. the repulsive forces generated by the terrain in a physical environment. We may also dynamically weight fields separately, e.g. 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 team-mate agents.

The third phase include to place the objects in the different fields. Static objects should perhaps be in the *field of navigation*. Typically, the potentials of such a field is pre-calculated in order to save precious run time CPU resources.

In *the fourth phase*, we have to decide the resolution of space and time. If the agents are able to move around in the environment, both these measures have an impact on the look-ahead. The space resolution, since it decides where in space we are able to go, and the time in that it determines how far we may get in one time frame.

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 look-ahead? 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).

3. ORTS

Open Real Time Strategy (ORTS) [4] 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 receive a data structure from the server containing the current game state. Clients can then issue commands for their units. Commands can be like move unit A to (x, y) or attack opponent unit X with unit A. All client commands are executed in random order by the server.

Users can define different type of games in scripts where units, structures and their interactions are described. All type of games from resource gathering to full real time strategy (RTS) games are supported. We focus on two types of two-player games, *tankbattle* and *tactical combat*. These games were part of the 2007 years ORTS competition [4].

- 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, i.e. 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 own bases with tanks. The map in a tankbattle game has randomly generated terrain with passable lowland and impassable cliffs.

- In *Tactical combat* each player has 50 marines and the goal is to destroy all the marines of the opponent. Marines have short fire range, average firepower and a short indestructible period. They are at the start of the game positioned randomly at either right or left side of the map. The map does not have any impassable cliffs.

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

4. MAPF IN ORTS

We have implemented an ORTS client for playing both Tankbattle and Tactical Combat based on MAPF following the proposed methodology. Below we will describe the creation of our MAPF solution.

4.1 Identifying objects

We identify the following objects in our applications: Cliffs, Sheep, and own (and opponent) tanks, marines and base stations.

4.2 Identifying fields

We identified four driving forces in ORTS: Avoid colliding with moving objects, Hunt down the enemy's forces and for the Tankbattle game also to Avoid colliding with cliffs, and to Defend the bases.

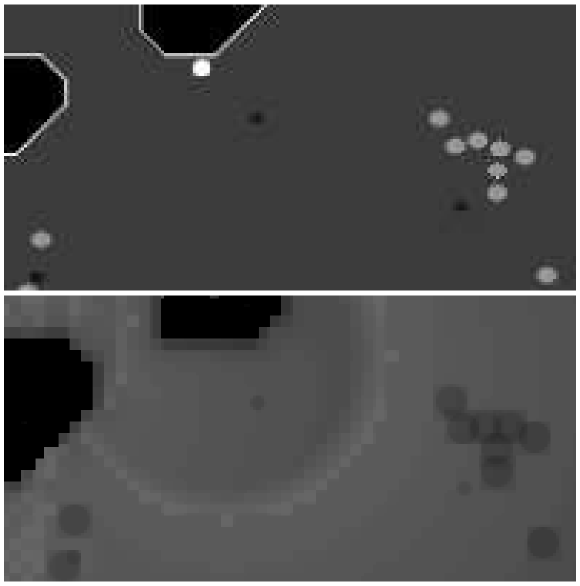

Figure 1: Part of the map during a tankbattle game. The upper picture shows our agents (light-grey circles), an opponent unit (white circle) and three sheep (small dark-grey circles). The lower picture shows the total potential field for the same area. Light areas has high potential and dark areas low potential.

This leads us to three types of potential fields: *Field of Navigation*, *Strategic Field*, and *Tactical field*.

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

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

Own units, own bases and sheep generate small repelling fields. The purpose is that we would like our agents to avoid colliding with each other or bases as well as avoiding the sheep.

4.3 Assigning charges

Each unit (own or enemy), control center, sheep and cliffs has a set of charges which generates a potential field around the object. Below you will find a more detailed description of the different fields. All fields generated by objects are weighted and summed to form a total field which is used by agents when selecting actions. The actual formulas for calculating the potentials very much depend on the application.

The upper picture in Figure 1 shows part of the map during a tankbattle game. The screen shot are from the 2D GUI available in the ORTS server. It shows our agents (light-grey circles) moving in to attack an 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 has low potential and light areas high potential. The light ring around the opponent unit, located at maximum shooting distance of our tanks, is the distance our agents prefer to attack opponent units from (see Section 4.5). It is the final move goal for our units. The picture also shows the small repelling field generated by own agents and sheep.

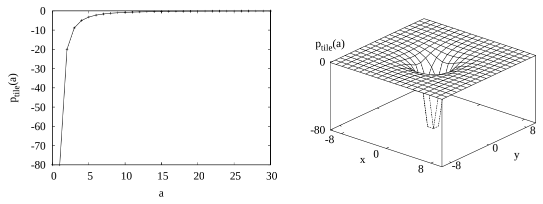

Figure 2: The potential $p_{cliff}(d)$ generated by a cliff given the distance d.

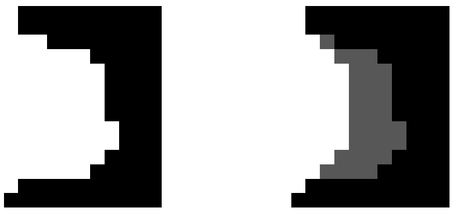

Figure 3: Example of the navigation field before and after filling dead ends. White are passable tiles, black impassable tiles and grey tiles filled by the algorithm.

Cliffs.

Cliffs generate a repelling field for obstacle avoidance. The potential $p_{cliff}(d)$ at distance d (in tiles) from a cliff is:

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

Note that if more than one cliff affects the same potential field tile, the actual potential is not calculated as the sum of the potentials (as in the other fields) but rather as the lowest value. This approach works better for passages between cliffs, see Figure 2.

The navigation field is post-processed in two steps to improve the agents abilities to move in narrow passages and avoid dead ends. The first step is to fill dead ends. The pseudo code below describes how this is done:

```
for all $x, y$ in navigation field $F(x, y)$ do
  if is_passable($x, y$) then
    $blocked = 0$
    for all 16 directions around $x, y$ do
      if cliff_within(5) then
        $blocked = blocked + 1$
      end if
    end for
    if $blocked >= 9$ then
      $IMPASSABLE(x, y) = true$
    end if
  end if
end for
```

For each passable tile (x, y) , we check if there are cliffs within 5 tiles in all 16 directions. If 9 or more directions are blocked by cliffs, we consider tile (x, y) impassable (Figure 3).

Next step is to clear narrow passages between cliffs from having a negative potential. This will make it easier for agents to use the passages, see Figure 4. Below is pseudo code for this processing step:

Unit	k_1	k_2	c_1	c_2	MSD	a	MDR
Marine	2	0.15	24.5	15	7	2	100
Tank	2	0.22	24.1	15	7	2	68
Base	3	0.255	49.1	15	12	2	130

Table 1: The parameters used for the generic $p(d)$-function of Eq. 2.

```
for all $x, y$ in navigation field $F(x, y)$ do
  $potential = p(x, y)$
  if $potential >= -50\ AND\ potential <= -1$ then
    if $p(x-1, y) < potential\ AND\ p(x+1, y) < potential$
    then
      $p(x, y) = 0$
    end if
    if $p(x, y-1) < potential\ AND\ p(x, y+1) < potential$
    then
      $p(x, y) = 0$
    end if
  end if
end for
```

For each passable tile (x, y) with negative potential, check if adjacent tiles has even lower negative potentials. If so, (x, y) is probably in a narrow passage and its potential is set to 0.

The opponent units.

All opponent units generates a symmetric surrounding field where the highest potential is in a ring around the object with a radius of MSD (*Maximum Shooting Distance*). As illustrated in Figure 5, MDR refers to the *Maximum Detection Range*, the distance from which an agent starts to detect the opponent unit. In general terms, the $p(d)$-function can be described as:

$$p(d) = \begin{cases} k_1 d, & \text{if } a \in [0, MSD - a[\\ c_1 - d, & \text{if } a \in [MSD - a, MSD] \\ c_2 - k_2 d, & \text{if } a \in]MSD, MDR] \end{cases} \tag{2}$$

Own bases.

Own bases generate a repelling field for obstacle avoidance. Below is the function for calculating the potential $p_{ownbase}(d)$ at distance d (in tiles) from the center of the base. Note that 4 is half the width of the base, and distances less than or equal to this value has a much lower potential. This approximation is not entirely correct at the corners of the base (since the base is quadratic rather than

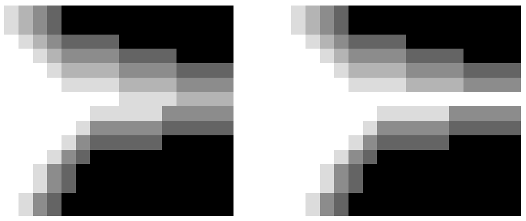

Figure 4: Example of the navigation field before and after clearing passages. White tiles has potential 0, and the darker the colour the more negative potential a tile has.

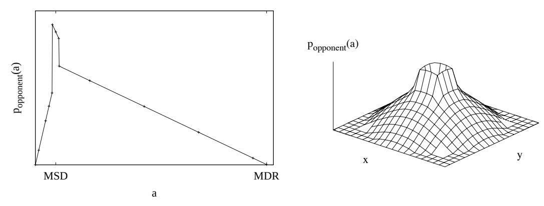

Figure 5: The potential $p_{opponent}(d)$ generated by the general opponent function given the distance d.

Unit	$radius$	k	c	l
Marine	0.5	3.2	5.6	1.75
Tank	0.875	3.2	10.8	3.375

Table 2: The parameters used for the generic $p_{ownunit}(d)$-function of Eq. 4.

circular, see Figure 6), but it works well in practice.

$$p_{ownbase}(d) = \begin{cases} 5.25 \cdot d - 37.5 & \text{if } d <= 4 \\ 3.5 \cdot d - 25 & \text{if } d \in]4, 7.14] \\ 0 & \text{if } d > 7.14 \end{cases} \tag{3}$$

The own mobile units — tanks and marines.

Own units, agents, generate a repelling field for obstacle avoidance (see Figure 7). In general terms, the potential $p_{ownunit}(d)$ at distance d (in tiles) from the center of an agent is calculated as:

$$p_{ownunit}(d) = \begin{cases} -20 & \text{if } d <= radius \\ d \cdot k - c & \text{if } d \in]radius, l], \\ 0 & \text{if } d >= l \end{cases} \tag{4}$$

Sheep.

Sheep generate a small repelling field for obstacle avoidance. The potential $p_{sheep}(d)$ (depicted in Figure 8) at distance d (in tiles) from the center of a sheep is calculated as:

$$p_{sheep}(d) = \begin{cases} -10 & \text{if } d <= 1 \\ -1 & \text{if } d \in]1, 2] \\ 0 & \text{if } d > 2 \end{cases} \tag{5}$$

4.4 On the granularity

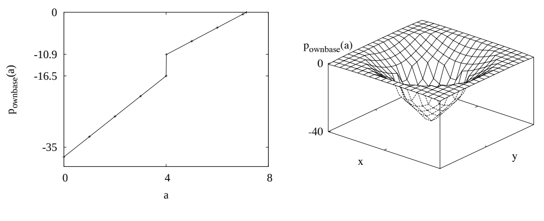

Figure 6: The repelling potential $p_{ownbase}(d)$ generated by the own bases given the distance d.

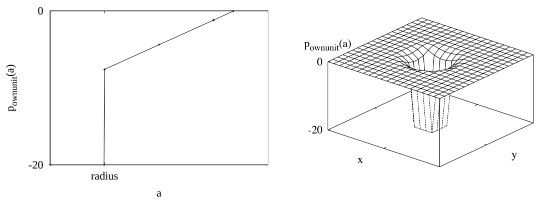

Figure 7: The repelling potential $p_{ownunit}(d)$ generated by the generic function given the distance d.

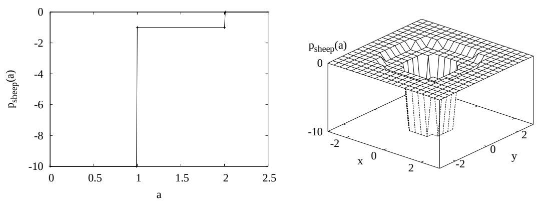

Figure 8: The potential $p_{sheep}(d)$ generated by a sheep given the distance d.

When designing the client we had to decide a resolution for the potential field. A tank-battle game has a map of 1024x1024 points and the terrain is constructed from tiles of 16x16 points. After some initial tests we decided to use 8x8 points for each tile in the potential field. The resolution had to be detailed enough for agents to be able to move around the game world using only the total potential field, but a more detailed resolution would have required more memory and the different fields would have been slower to update.[1] Thus in our implementation 8x8 points was found to be a good trade-off.

4.5 The unit agent(s)

When deciding actions for an agent, the potential of the tile the agent is at is compared with the potentials of the surrounding tiles. The agent moves to the center of the neighbour tile with the highest potential, or is idle if the current tile is highest. If an agent has been idle for some time, it moves some distance in a random direction to avoid getting stuck in a local maxima. If an opponent unit is within fire range, the agent stops to attack the enemy.

Since there is an advantage of keeping the agents close to the maximum shooting distance (MSD), the positions of the opponent units are not the final goal of navigation. Instead we would like to keep them near the MSD. The obstacles should be avoided, roughly in the sense that the further away they are, the better it is. Here, the own agents are considered to be obstacles (for the ability to move).

When an agent executes a move action, the tactical field is updated with a negative potential (same as the potential around own agents) at the agents destination. This prevents other agents from moving to the same position if there are other routes available.

4.6 The MAS architecture

In a tank-battle game our agents has two high-level tactical goals. If we have a numerical advantage over the opponent units we attack both bases and units. If not, we attack units only and wait with attacking bases. For agents to attack both units and bases, one of the following constraints must be fulfilled:

[1]The number of positions quadruples as the resolution doubles.

- We must have at least twice as many tanks as the opponent
- The opponent have less than six tanks left
- The opponent have only one base left

If none of these constraints are fulfilled, the tactical goal is to attack opponent units only. In this case the field generated by opponent bases are not an attracting field. Instead they generate a repelling field for obstacle avoidance (same as the field generated by own bases). We want to prevent our agents from colliding with opponent bases if their goal is not to attack them. In a tactical combat game no bases are present and agents always aim to destroy opponent marines.

4.6.1 Attack coordination

We use a coordinator agent to globally optimise attacks at opponent units. The coordinator aims to destroy as many opponent units as possible each frame by concentrating fire on already damaged units. Below is a description of how the coordinator agent works. After the coordinator is finished we have a near-optimal allocation of which of our agents that are dedicated to attack which opponent units or bases.

The coordinator uses an attack possibility matrix. The $i \times k$ matrix A defines the opponent units i (out of n) within MSD which can be attacked by our agents k (out of m) as follows:

$$a_{k,i} = \begin{cases} 1 & \text{if the agent } k \text{ can attack opponent unit } i \\ 0 & \text{if the agent } k \text{ cannot attack opponent unit } i \end{cases} \tag{6}$$

$$A = \begin{bmatrix} a_{0,0} & \cdots & a_{m-1,0} \\ \vdots & \ddots & \vdots \\ a_{0,n-1} & \cdots & a_{m-1,n-1} \end{bmatrix} \tag{7}$$

We also need to keep track of current hit points (HP) of the opponent units i as:

$$HP = \begin{bmatrix} HP_0 \\ \vdots \\ HP_{n-1} \end{bmatrix} \tag{8}$$

Let us follow the example below to see how the coordination heuristic works.

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad HP = \begin{bmatrix} HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_3 = 4 \\ HP_4 = 4 \\ HP_5 = 3 \end{bmatrix} \tag{9}$$

First we sort the rows so the highest priority targets (units with low HP) are in the top rows. This is how the example matrix looks like after sorting:

$$A_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad HP = \begin{bmatrix} HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_5 = 3 \\ HP_4 = 4 \\ HP_3 = 4 \end{bmatrix} \tag{10}$$

Next step is to find opponent units that can be destroyed this frame (i.e. we have enough agents able to attack an opponent unit to reduce its HP to 0). In the example we have enough agents within range to destroy unit 0 and 1. We must also make sure that the agents attacking unit 0 or 1 are not attacking other opponent units in A. This is done by assigning a 0 value to the rest of the column in A for all agents attacking unit 0 or 1.

Below is the updated example matrix. Note that we have left out some elements for clarity. These has not been altered in this step and are the same as in matrix A_2.

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & & & 0 & \\ 0 & 0 & 0 & 0 & & & 0 & \\ 0 & 0 & 0 & 0 & & & 0 & \\ 0 & 0 & 0 & 0 & & & 0 & \end{bmatrix} \quad HP = \begin{bmatrix} HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_5 = 3 \\ HP_4 = 4 \\ HP_3 = 4 \end{bmatrix} \tag{11}$$

The final step is to make sure the agents in the remaining rows (3 to 6) only attacks one opponent unit each. This is done by, as in the previous step, selecting a target i for each agent (start with row 3 and process each row in ascending order) and assign a 0 to the rest of the column in A for the agent attacking i. This is how the example matrix looks like after the coordinator is finished:

$$A_4 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad HP = \begin{bmatrix} HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_5 = 3 \\ HP_4 = 4 \\ HP_3 = 4 \end{bmatrix} \tag{12}$$

In the example the fire coordinator agent have optimised attacks to:

- Unit 0 is attacked by agents 0 and 3. It should be destroyed.
- Unit 1 is attacked by agents 1, 2 and 6. It should be destroyed.
- Unit 5 is attacked by agent 6. Its HP should be reduced to 2.
- Unit 4 is attacked by agents 4 and 5. Its HP should be reduced to 2.
- Units 2 and 3 are not attacked by any agent.

4.6.2 *The Internals of the Coordinator Agent*

The coordinator agent first receive information from each of the own agents. It contains its positions and ready-status, as well as a list of the opponent units that are within range. Ready-status means that an agent is ready to fire at enemies. After an attack a unit has a cool-down period while it cannot fire. From the server, it will get the current hit point status of the opponent units.

Now, the coordinator filters the agent information so that only those agents that are i) ready to fire and ii). have at least one opponent unit within MSD, are left.

For each agent k that is ready to fire, we iterate through all opponent units and bases. To see if k can attack unit i we use a three level check:

1. Agent k must be within Manhattan distance[2] * 2 of i (very fast but inaccurate calculation)
2. Agent k must be within real (Euclidean) distance of i (slower but accurate calculation)
3. Opponent unit i must be in line of sight of k (very slow but necessary to detect obstacles in front of i)

The motivation behind the three-level check is to start with fast but inaccurate calculations, and for each level passed a slower and more accurate check is performed. This reduces CPU usage by skipping demanding calculations such as line-of-sight for opponent units or bases that are far away.

Next step is to sort the rows in A in ascending order based on their HP (prioritise attacking damaged units). If two opponent units has same hit points left, the unit i which can be attacked by the largest number of agents k should be first (i.e. concentrate fire to damage a single unit as much as possible rather than spreading the fire). When an agent attacks an opponent unit it deals a damage value randomly chosen between the attacking unit's minimum (min_{dmg}) and maximum (max_{dmg}) damage. A unit hit by an attack get its HP reduced by the damage value of the attacking unit minus its own armour value. The armour value is static and a unit's armour cannot be destroyed.

The next step is to find opponent units which can be destroyed this frame. For every opponent unit i in A, check if enough agents u can attack i to destroy it as:

$$\left(\sum_{k=0}^{m-1} a(k,i)\right) \cdot (damage_u - armour_i) >= HP_i \tag{13}$$

$armour_i$ is the armour value for the unit type of i (0 for marines and bases, 1 for tanks) and $damage_u = min_{dmg} + p \cdot (max_{dmg} - min_{dmg})$, where $p \in [0, 1]$. We have used a p value of 0.75, but it can be changed to alter the possibility of actually destroying opponent units.

If more agents can attack i than is necessary to destroy it, remove the agents with the most occurrences in A from attacking i. The motivation behind this is that the agents u with most occurrences in A has more options when attacking other units.

At last we must make sure the agents attacking i does not attack other opponent units in A. This is done by assigning a 0 value to the rest of the column.

The final step is to make sure agents not processed in the previous step only attacks one opponent unit each. Iterate through every i that cannot be destroyed but can be attacked by at least one agent k, and assign a 0 value to the rest of the column for each k attacking i.

5. EXPERIMENTS

Our bot have participated in the 2007 years ORTS competition. Below is a brief description of the other competition entries [4]. The results from the competition are presented in Tables 3–4. As we can see from the results summary our bot was not among the top entries in the competition, but rather in the bottom half. We did however win almost a third of the played games in both categories. Note that all other competition entries are based on more traditional approaches with pathfinding and higher level planning, and our goal is to investigate if our Multi-agent Potential Fields based bot is able to reach the same level of performance as the traditional solutions.

The team *NUS* use finite state machines and influence maps in high-order planning on group level. The units in a squad spread out on a line and surround the opponent units at MSD. Units use the cool-down period to keep out of MSD. Pathfinding and a flocking algorithm is used to avoid collisions.

UBC gather units in squads of 10 tanks or marines. Squads can be merged with other squads or split into two during the game. Pathfinding is combined with force fields to avoid obstacles and bit-mask for collision avoidance. Units spread out at MSD when attacking. Weaker squads are assigned to weak spots or corners of the opponent unit cluster. If an own base is attacked, it may decide to try to defend the base.

[2]The Manhattan distance between two coordinates $(x_1, y_1), (x_2, y_2)$ is given by $abs(x_1 - x_2) + abs(y_1 - y_2)$.

Team	Wins ratio	Wins/games	Team name
nus	98%	(315/320)	National Univ. of Singapore
WarsawB	78%	(251/320)	Warsaw Univ., Poland
ubc	75%	(241/320)	Univ. of British Columbia, Canada
uofa	64%	(205/320)	Univ. of Alberta, Canada
uofa.06	46%	(148/320)	Univ. of Alberta
BTH	**32%**	**(102.5/320)**	**Blekinge Inst. of Tech., Sweden**
WarsawA	30%	(98.5/320)	Warsaw University, Poland
umaas.06	18%	(59/320)	Univ. of Maastricht, The Netherlands
umich	6%	(20/320)	Univ. of Michigan, USA

Table 3: Summary of the results of ORTS tank-battle 2007

Team	Wins ratio	Wins/games	Team name
nus	99%	(693/700)	National Univ. of Singapore
ubc	75%	(525/700)	Univ. of British Columbia, Canada
WarsawB	64%	(451/700)	Warsaw Univ., Poland
WarsawA	63%	(443/700)	Warsaw Univ., Poland
uofa	55%	(386/700)	Univ. of Alberta, Canada
BTH	**28%**	**(198/700)**	**Blekinge Inst. of Tech., Sweden**
nps	15%	(102/700)	Naval Postgraduate School, USA
umich	0%	(2/700)	Univ. of Michigan, USA

Table 4: Summary of the results of the ORTS tactical combat

WarsawA synchronises units by assigning each unit position to a node in a grid. The grid is also used for pathfinding. When units are synchronised they attack the enemy at a line going for its weakest spots at a predefined distance.

WarsawB uses pathfinding with an additional dynamic graph for moving objects. Own units uses a repelling force field collision avoidance. Units are gathered in one large squad. When the squad attacks, its units spread out on a line at MSD and each unit attack the weakest opponent unit in range. In tactical combat, each own unit is assigned to an opponent unit and it always tries to be at the same horizontal line (y coordinate) as its assigned unit.

Uofa uses a hierarchical commander approach ranging from squad commanders down to pathfinding and attack coordination commanders. Units are grouped in a single, large cluster and tries to surround the opponent units by spreading out at MSD. The hierarchical commander approach is not used in tactical combat.

Umich uses an approach where the overall tactics are implemented in the SOAR language. SOAR in turn have access to low-level finite state machines for handling, for example, squad movement. Units are gathered in a single squad hunting enemies, and opponent units attacking own bases are the primary goals.

Umaas and *Uofa* entered the competition with their 2006 years entries. No entry descriptions are available.

6. DISCUSSION

We discuss potential fields in general, then the results of the experiments, and finally write a few words about the methodology.

6.1 The use of PF in games

Traditionally the use of potential fields (PF), although having gained some success in the area of robotic navigation, has been limited in the domain of game AI. There are a number of more or less good reasons for that:

1. PF are considered to be less controllable than traditional planning [16]. This may be an important feature in the early stages of a game development.

2. A* and different domain specific improvements of it has proven to gain sufficiently good results.

3. PF based methods are believed to be hard to implement and to debug. These problems may especially apply to the representation of the environment, and the dynamic stability [16].

4. Agents navigating using PFs often get stuck in local optima.

However, from the reported use of potential fields in the area of RoboCup and games indicate that:

PF may be implemented in a way that use the processing time efficiently, especially in highly dynamic environments where lots of objects are moving and long term planning is intractable. By just focusing on nine options (eight directions + standing still) we do, at most, have to calculate the potentials of $9n$ positions for our n units. All potential functions may be pre-calculated and stored in arrays, which makes the actual calculation of the potential of a position just a matter of summing up a number of array elements.

By using multiple maps over the potential landscape (e.g. one for each type of unit), the debug process becomes significantly more efficient. We used different potential landscapes that were put on the map to illustrate the potentials using different colours.

The great thing with PFs is that the attracting – repelling paradigm is very intuitive: the good outcomes of actions are attractive, and the bad outcomes repellent. Thus an action that lead to both bad and good outcomes can be tuned at the outcome level, rather than on the action level.

In static environments, the local optima problem has to be dealt with when using PF. In ORTS, which in some cases is surprisingly static, we used convex filling and path clearing of the terrain to help the units, but this did not always help. We believe that more efforts here will improve the performance. Thurau et al. [15] describes a solution to the local maxima problem called avoid-past potential field forces. Each of their agents generate a trail of negative potential, similar to a pheromone trail used by ants, at visited positions. The trail pushes the agent forward if it reaches a local maximum. This approach may work for our agents as well.

6.2 The Experiments

There are a number of possible explanations for the good results of the top teams (and the comparative bad results for our team). First, the top teams are very good at handling difficult terrain which, since the terrain is generated randomly, sometimes cause problems for our agents due to local optima.

The second advantage is coordinating units in well-formed squads. Since we do not have any attracting mechanism between agents and higher-level grouping of squads, our agents are often spread out with a large distance between them. Enemies can in some cases destroy our agents one at a time without risk of being attacked by a large number of coordinated agents.

The third advantage is that the top teams spread out units at MSD, and always tries to keep that distance. Since the field of opponents are a sum of the generated potentials for all opponent units, the maxima tend to be in the center of the opponent cluster and our agents therefore attack the enemy at their strongest locations instead of surrounding the enemy.

We believe it is possible to solve these issues using MAPF. The first issue is a matter of details in the resolution of the MAPF. Our agents move to the center of the 8x8 points tile with highest potential. This does not work very well for narrow passages or if bases, other agents or sheep are close. This could be solved by either increasing the resolution of the MAPF or add functionality for estimating a potential at a point to enable movement at point level.

The second issue can be solved by using a both positive and negative field for agents. Close to the agents, there is a surrounding negative field as in our implementation, which in turn is surrounded by a positive one. The positive field will make the agents to keep an appropriate distance and possibly having an emergent effect of surrounding the opponent (see e.g. Mamei and Zambonelli [11]).

The third issue can be solved by not calculating the potential in a point as the sum of the potentials all opponent units generate in that point, but rather the highest potential an opponent unit generate in the point. This will make sure the maxima in the strategic field always are at MSD even if the opponent units are clustered in large groups, and our agents will more likely surround the enemy.

To further improve our bot a new type of tactics field can be used. By generating a large positive field at the weakest spot of the opponent units cluster, agents attack the weakest spot instead of attacking strong locations. This field differs from the other fields used in that it is not generated by a game object, but rather generated by a higher-level tactical decision.

6.3 On the Methodology

We chose to implement and test our idea of using a Multi-agent Potential Field based solution in the yearly ORTS competition. As a testbed, we believe that it is good for this purpose for a number of reasons: i). It is a competition, meaning that others will do their best to beat us. ii) It provides a standardised way of benchmarking Game AI solutions iii). The environment is open source and all of the mechanics are transparent. iv) ORTS uses a client-server architecture where clients only has access to the information sent by the server. No client can gain an advantage by hacking the game engine as often is possible in a peer-to-peer architecture. v) Even though ORTS is written in C++ the communication protocol is public and it is possible to write a wrapper to any other language. The results may seem modest, but we show that MAPFs is an alternative to A* based solutions in the case of ORTS. We have no reason to believe that MAPF would not be successful in other RTS games.

7. CONCLUSIONS AND FUTURE WORK

A long-term plan, for example path finding, generated by an agent might need re-planning if the game world changes during the execution of the plan. With a PF based solution path planning may be replaced by one step look-ahead, if the analysis is carried out carefully, but yet efficiently. We believe that in ORTS, MAPFs fulfils the requirements of efficiency and flexibility and conclude that MAPF is indeed an interesting alternative worth investigating further. However, more research is needed on how to implement MAPF based solutions in general, and on what tools to use in the debugging and calibration process. Preliminary late results show that our MAPF solution now beat all the competitors of the 2007 ORTS competition. The future of MAPF looks bright and we hope to be able to report further on this in the near future. Future work include to optimise the parameters using e.g. genetic algorithms, to take care of the issues mentioned in Section 6, and to refine the agent perspective through distributing the coordination of attacks and the exploration of the map explicitly. We would also like to try our approach in other domains.

8. REFERENCES

[1] R. C. Arkin. Motor schema based navigation for a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 264–271, 1987.

[2] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19:1179–1187, 1989.

[3] J. Borenstein and Y. Koren. The vector field histogram: fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, 1991.

[4] M. Buro. ORTS — A Free Software RTS Game Engine, 2007. http://www.cs.ualberta.ca/ mburo/orts/ URL last visited on 2008-01-25.

[5] 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 (DARS02)*, 2002.

[6] S. Johansson. On using multi-agent systems in playing board games. In *Proceedings of Autonomous Agents and Multi-agent Systems (*AAMAS*)*, 2006.

[7] S. Johansson and A. Saffiotti. An electric field approach to autonomous robot control. In *RoboCup 2001*, number 2752 in Lecture notes in artificial intelligence. Springer Verlag, 2002.

[8] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[9] O. Khatib. Human-like motion from physiologically-based potential energies. In J. Lenarcic and C. Galletti, editors, *On Advances in Robot Kinematics*, pages 149–163. Kluwer Academic Publishers, 2004.

[10] S. Kraus and D. Lehmann. Designing and building a negotiating automated agent. *Computational Intelligence*, 11(1):132–171, 1995.

[11] M. Mamei and F. Zambonelli. Motion coordination in the quake 3 arena environment: A field-based approach. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*. IEEE Computer Society, 2004.

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

[13] C. Niederberger and M. H. Gross. Towards a game agent. Technical Report 377, Swiss Federal Institute of Technology, Zürich, 2003.

[14] T. Röfer, R. Brunn, I. Dahm, M. Hebbel, J. Homann, M. Jüngel, T. Laue, M. Lötzsch, W. Nistico, and M. Spranger. GermanTeam 2004 - the german national Robocup team, 2004.

[15] C. Thurau, C. Bauckhage, and G. Sagerer. Learning human-like movement behavior for computer games. In *Proc. 8th Int. Conf. on the Simulation of Adaptive Behavior (SAB'04)*, 2004.

[16] S. L. Tomlinson. The long and short of steering in games. *International Journal of Simulations*, 1–2(5), 2004.

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

[18] M. van Lent, J. Laird, J. Buckman, J. Hartford, S. Houchard, K. Steinkraus, and R. Tedrake. Intelligent agents in computer games. In *Proceedings of AAAI*. AAAI, 1999.