

The Rise of Potential Fields in Real Time Strategy Bots

Johan Hagelbäck and **Stefan J. Johansson**

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

Abstract

Bots for Real Time Strategy (RTS) games are challenging 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. We show that the use of potential fields for implementing a bot for a real time strategy game gives us a very competitive, configurable, and non-conventional solution.

Keywords

Agents:Swarm Intelligence and Emergent Behavior, Multi-disciplinary Topics and Applications:Computer Games

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 their opponents, 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 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 (Khatib 1986). Later on Arkin (Arkin 1987) 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.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Many studies concerning potential fields are related to spatial navigation and obstacle avoidance, see e.g. (Borenstein & Koren 1991; Massari, Giardini, & Bernelli-Zazzera 2004). 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 (Borenstein & Koren 1989).

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 (Howard, Matarić, & Sukhatme 2002), and potential fields have been used in robot soccer (Johansson & Saffiotti 2002; Röfer *et al.* 2004). Thureau et al. (Thureau, Bauckhage, & Sagerer 2004b) 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.

First we describe the domain followed by a description of our basic MAPF player. That solution is refined stepwise in a number of ways and for each and one of them we present the improvement shown in the results of the experiments. We then discuss the solution and conclude and show some directions of future work. We have previously reported on the details of our methodology, and made a comparison of the computational costs of the bots, thus we refer to that study for these results (Hagelbäck & Johansson 2008).

ORTS

Open Real Time Strategy (ORTS) (Buro 2007) 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 time-frame clients receive a data structure from the server containing the current game state. Clients can then issue commands for their units. Commands such as 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 strat-

egy (RTS) games are supported. We focus here on one type of two-player game, *Tankbattle*, which was one of the 2007 ORTS competitions (Buro 2007). 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.

The game contains a number of neutral units (sheep). These are small indestructible units moving randomly around the map making pathfinding and collision detection more complex.

The Tankbattle competition of 2007

For comparison, the results from our original bot against the four top teams were reconstructed through running the matches again (see Table 1). To get a more detailed comparison than the win/lose ratio used in the tournament we introduce a game score. This score does not take wins or losses into consideration, instead it counts units and bases left after a game. The score for a game is calculated as:

$$score = 5(ownBasesLeft - oppBasesLeft) + ownUnitsLeft - oppUnitsLeft \quad (1)$$

Opponent descriptions

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

UBC gathers units in squads of 10 tanks. Squads can be merged with other squads or split into two during the game. Pathfinding is combined with force fields to avoid obstacles and a 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.

WarsawB uses pathfinding with an additional dynamic graph for moving objects. The units use 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 attack the weakest opponent unit in range.

Uofa06 Unfortunately, we have no description of how this bot works, more than that it was the winner of the 2006 year ORTS competition. Since we failed in getting the 2007 version of the UofA bot to run without stability problems under the latest update of the ORTS environment, we omitted it from our experiments.

MAPF in ORTS, V.1

We have implemented an ORTS client for playing Tankbattle based on Multi-agent Potential Fields (MAPF) following the

Team	Win %	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

Table 1: Replication of the results of our bot in the ORTS tournament 2007 using the latest version of the ORTS server.

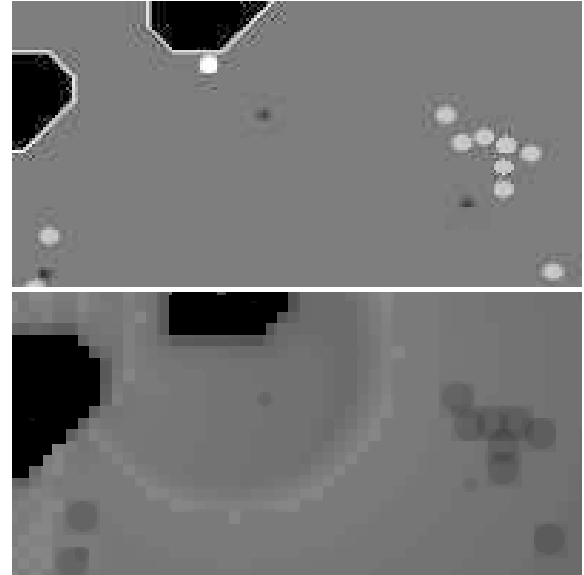


Figure 1: Part of the map during a tankbattle game. The upper picture shows our agents (light-grey circles), an opponent unit (white circle) and three sheep (small dark-grey circles). The lower picture shows the total potential field for the same area. Light areas has high potential and dark areas low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

proposed methodology of Hagelbäck and Johansson (Hagelbäck & Johansson 2008). It includes the following six steps:

1. Identifying the objects
2. Identifying the fields
3. Assigning the charges
4. Deciding on the granularities
5. Agentifying the core objects
6. Construct the MAS architecture

Below we will describe the creation of our MAPF solution.

Identifying objects

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

Identifying fields

We identified four tasks in ORTS Tankbattle: Avoid colliding with moving objects, Hunt down the enemy’s forces, Avoid

colliding with cliffs, and Defend the bases. 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 and may be pre-calculated in the initialisation phase. 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 at appropriate distances from where they can fight the enemies.

Our 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.

Assigning charges

Each unit (own or enemy), base, sheep and cliff have a set of charges which generate a potential field around the object. All fields generated by objects are weighted and summed to form a total field which is used by agents when selecting actions. The initial set of charges were found using trial and error. However, the order of importance between the objects simplifies the process of finding good values and the method seems robust enough to allow the bot to work good anyhow. We have tried to use traditional AI methods such as genetic algorithms to tune the parameters of the bot, but without success. The results of these studies are still unpublished. We used the following charges in the V.1 bot:¹

The opponent units

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

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

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

Own bases Own bases generate a repelling field for obstacle avoidance. Below in Equation 3 is the function for calculating the potential $p_{ownB}(d)$ at distance d (in tiles) from the center of the base.

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

¹ $I = [a, b[$ denote the half-open interval where $a \in I$, but $b \notin I$

The own tanks The potential $p_{ownU}(d)$ at distance d (in tiles) from the center of an own tank is calculated as:

$$p_{ownU}(d) = \begin{cases} -20 & \text{if } d \leq 0.875 \\ 3.2d - 10.8 & \text{if } d \in]0.875, l[\\ 0 & \text{if } d \geq l \end{cases} \quad (4)$$

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

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

Figure 1 shows an example of a part of the map during a Tankbattle game. The screen shot are from the 2D GUI available in the ORTS server, and from our own interface for showing the potential fields. 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. The picture also shows the small repelling fields generated by our own units and the sheep.

Granularity

We believed that tiles of 8*8 positions was a good balance between performance on the one hand, and the time it would take to make the calculations, on the other.

Agentifying and the construction of the MAS

We put one agent in each unit, and added a coordinator that took care of the coordination of fire. For details on the implementation description we have followed, we refer to Hagelbäck and Johansson (Hagelbäck & Johansson 2008).

Weaknesses and counter-strategies

To improve the performance of our bot we observed how it behaved against the top teams from the 2007 years' ORTS tournament. From the observations we have defined a number of weaknesses of our bot and proposed solutions to these. For each improvement we have run 100 games against each of the teams NUS, WarsawB, UBC and Uofa.06. A short description of the opponent bots can be found below. The experiments are started with a randomly generated seed and then two games, one where our bot is team 0 and one where our bot is team 1, are played. For the next two games the seed is incremented by 1, and the experiments continues in this fashion until 100 games are played.

By studying the matches, we identified four problems with our solution:

1. Some of our units got stuck in the terrain due to problems finding their way through narrow passages.
2. Our units exposed themselves to hostile fire during the cool down phase.
3. Some of the units were not able to get out of local minima created by the potential field.

