A Multi-Agent Potential Field Based Approach for Real-Time Strategy Game Bots

Johan Hagelbäck

Department of Systems and Software Engineering
School of Engineering
Publisher: Blekinge Institute of Technology
Printed by Printfabriken, Karlskrona, Sweden 2009
ISBN 978-91-7295-160-0

Blekinge Institute of Technology
Licentiate Dissertation Series No. 2009:03
ISSN 1650-2140
ISBN 978-91-7295-160-0

A Multi-Agent Potential Field Based Approach for Real-Time Strategy Game Bots

Johan Hagelbäck

Department of Systems and Software Engineering
School of Engineering
Blekinge Institute of Technology
Sweden

Contact information:
Johan Hagelbäck
Department of Systems and Software Engineering
School of Engineering
Blekinge Institute of Technology
P.O. Box 520
372 25 Ronneby
SWEDEN
email: Johan.Hagelback@bth.se

"In preparing for battle I have always found that plans are useless, but planning is indispensable."

– Dwight D. Eisenhower

ABSTRACT

Computer games in general and Real-Time Strategy (RTS) games in particular provide a rich challenge for both human- and coimputer controlled players, often denoted as bots. The player or bot controls a large number of units that have to navigate in partially unknown dynamic worlds to pursue a goal. Navigation in such worlds can be complex and require much computational resources. Typically it is solved by using some sort of path planning algorithm, and a lot of research has been conducted to improve the performance of such algorithms in dynamic worlds. The main goal of this thesis is to investigate an alternative approach for RTS bots based on Artificial Potential Fields, an area originating from robotics. In robotics the technique has successfully been used for navigation in dynamic environments, and we show that it is possible to use Artificial Potential Fields for navigation in an RTS game setting without any need of path planning.

In the first three papers we define and demonstrate a methodology for creating multi-agent potential field based bots for an RTS game scenario where two tank armies battle each other. The fourth paper addresses incomplete information about the game world, referred to as the fog of war, and show how Potential Field based bots can handle such environments. The final paper shows how a Potential Field based bot can be evolved to handle a more complex full RTS scenario. It addresses resource gathering, construction of bases, technological development and construction of an army consisting of different types of units.

We show that Artificial Potential Fields is a viable option for several RTS game scenarios and that the performance, both in terms of being able to win a game and computational resources used, can match and even surpass those of traditional approaches based on path planning.

ACKNOWLEDGMENTS

First, I would like to thank my main supervisor Dr. Stefan J. Johansson for invaluable support and guidance throughout the work. I also would like to thank my secondary supervisors Professor Paul Davidsson and Professor Craig Lindley for encouraging my work, and the members of the DISL research group for valuable comments and ideas. In addition I would like to thank Dr. Michael Buro at University of Alberta for support in using the ORTS platform.

Last but not least I would like to thank my wife Maria for always being there for me. This thesis would not have been possible without your support.

Ronneby, March 2009
Johan Hagelbäck

PREFACE

This thesis is a compilation of five papers. The papers are listed below and will be referenced to in the text by the associated Roman numerals. The previously published papers have been reformatted to suit the thesis template.

I. J. Hagelbäck and S. J. Johansson (2008). Using Multi-agent Potential Fields in Real-time Strategy Games. In L. Padgham and D. Parkes editors, Proceedings of the Seventh International Conference on Autonomous Agents and Multi-agent Systems (AAMAS).

II. J. Hagelbäck and S. J. Johansson (2008). Demonstration of Multi-agent Potential Fields in Real-time Strategy Games. Demo Paper on the Seventh International Conference on Autonomous Agents and Multi-agent Systems (AAMAS).

III. J. Hagelbäck and S. J. Johansson (2008). The Rise of Potential Fields in Real Time Strategy Bots. In Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE).

IV. J. Hagelbäck and S. J. Johansson (2008). Dealing with Fog of War in a Real Time Strategy Game Environment. In Proceedings of 2008 IEEE Symposium on Computational Intelligence and Games (CIG).

V. J. Hagelbäck and S. J. Johansson. A Multi-agent Architecture for Real Time Strategy Games. Submitted for publication.

The author of the thesis is the main contributor to all of these papers. In addition, the following paper is related to the thesis:

VI. J. Hagelbäck and S. J. Johansson (2009), A Multiagent Potential Field-Based Bot for Real-Time Strategy Games. International Journal of Computer Games Technology, vol. 2009, Article ID 910819, 10 pages. doi:10.1155/2009/910819

Papers VI is a summary with some additions of the work presented in papers I, II, III and IV thus it is not included in the thesis.

CONTENTS

CHAPTER

ONE

INTRODUCTION

A Real-Time Strategy (RTS) game is a game that lets players simulate resource gathering, construction of bases, technological development and unit control in order to defeat their opponent(s). The game is typically set in a war scenario in which the player controls the construction and actions of an army. The game environment can range from medieval (*Age of Empires 2*), fantasy (*Warcraft II* and *III*), World War II (*Commandos I* and *II*), modern (*Command & Conquer Generals*) to science fiction (*Starcraft* and *Dawn of War*). The game is running in real-time in contrast to turn-based board games such as *Risk* and *Diplomacy*.

From a general perspective, an RTS game consists of the components listed below (Buro & Furtak, 2004; Co, 2007).

- *Workers*. Workers are used to gather resources which must be spent wisely on creating new workers, combat units such as tanks and marines, construction of buildings and technological development. Typically the worker has to move to an area containing resources (for example a mine), spend some time gathering as much resources as it can carry, and return to a base to drop them off. In many games workers have to gather several types of resources, for example both lumber and gold in *Warcraft III*. Workers are also often needed to construct new buildings.

- *Control centers*. A player often has one, sometimes more, bases to control. The control center is the core of a base. A base is built around the control center and no other buildings can be constructed without it. New workers are typically constructed in a control center, and the control center is often also the place where workers drop off gathered resources. A game is often won when all the control centers of the opponent(s) are destroyed.

- *Combat units*. Combat units such as marines and tanks are used to explore the game world, defend own bases and attack and defeat enemy forces. A game often has a wide range of units with different strengths and weaknesses. Tanks can be very strong attacking units, siege tanks can have devastating long ranged firepower but has to be stationary to be able to fire, and marines are mobile but weak and can carry grenades which are effective against other marines. When designing units for a game it is important to avoid the "tank race" problem. This means that there is a single over-powered unit that is the key to success, and the first player able to mass construct that unit is quite certain to win the game. This was for example the case in *Command & Conquer Generals*, where the chinese had access to a large tank with strong cannons effective against vehicles and buildings and mounted machine guns effective against foot soldiers. If the players control different races or factions, for example orcs and humans in *Warcraft II*, it is important to avoid unbalance between them so no player is favored due to choosing a race or faction which has benefits over the others and are more likely to win a game.

- *Barracks and factories*. Barracks are buildings used to train marines, and factories are buildings used to construct vehicles such as tanks and artillery. The cost of constructing units usually differs based on the relative strength of the units. Usually a player has to construct a barrack before gaining access to factories which in turn can produce stronger units.

- *Additional structures*. Barracks and factories are, with some variants, present in the majority of RTS games. To give the player(s) more possibilities and enrichen the gameplay additional structures are often available. Some examples are walls to defend the bases, stationary defensive towers with ranged weapons, radars that increase the visibility range of the own bases, silos that increase the rate in which resources are gathered, and much more.

In addition, RTS games often have *Technology trees*. The more powerful units and structures are often not immediately available to a player when the game begins. For example, the player has to construct a barrack before he/she is able to construct a factory. If a factory and a radar building has been constructed, an airfield can be built to gain access to fighter and helicopter units. The player has to decide whether to construct cheap but less powerful units such as marines, or to spend resources on buildings in order to unlock more powerful units such as tanks. This allows for some different tactics, such as *soldier rush*, to overwhelm the enemy early on in a game by a large number of marines, or to play defensively until a powerful force based on tanks and support units has been produced.

The endless possibilities, the richness and complexity of the game world makes the design and development of computer players, often denoted as bots, for RTS games challenging. The bot has to control an often very large number of units that have to

navigate in the game world, it has to make complex tactical decisions and the real-time aspect makes it even more challenging. It is important that the bot is robust, that the controlled units make reasonably realistic actions, and that it is fast enough to avoid getting in the situation where units stand idle because the bot is *thinking*.

A typical RTS game has scenarios with different goals and setups. Two possible scenarios are:

- *Tankbattle*. Tankbattle is a two player game where each player starts with one or more bases and a number of combat units spread out in the game world. The first player that manage to destroy the base of the opponent wins the game. Players are not able to produce more combat units or bases during the game.

- *Full RTS*. Full RTS is a more complex scenario where each player starts with a control center, a number of workers and maybe some combat unit. A player must use the workers to gather resources and to construct new control centers, barracks and factories. In addition resources must be spent on constructing mobile combat units, which in turn are used to attack the enemy and defend own bases. The first player to destroy all buildings of the opponent wins the game.

In this thesis we will investigate an alternative approach for navigation and path planning in an RTS game. The approach, based on Artificial Potential Fields, will be evaluated in the two scenarios Tankbattle and Full RTS.

1.1 Background and Related Work

A bot for a real-time strategy game is often constructed using a structured hierarchy of layers with more and more specific responsibilities. Reynolds (2002) proposed a layered architecture with four commanders: *Soldier*, *Sergeant*, *Captain* and *Commander*. Each level has different responsibilities, with lower levels being subordinate to higher levels. The Commander is responsible for the overall strategical decisions and issues orders to the Captains. Each Captain in turn has control over a number of squads and resources which he can distribute to execute the orders from the Commander. The Sergeant is typically the leader of a squad, a group of combat units, which he uses in order to execute the orders from the Captain. Finally, there is the Soldier who is responsible for controlling and navigating a single unit in the game world, report on the status of activities to the Sergeant, and report on changes in the game world (e.g. a newly discovered threat).

Navigating a unit in the game world is a complex task. The unit has to find a path to its destination through the terrain, avoid colliding with obstacles and re-plan if the current path becomes obsolete due to changes in the often highly dynamic game world. Combat units must also engage the enemy in a coordinated manner since spread out units are easy preys for the enemy. The path planning system must be effective enough to allow a large number of units to move concurrently and collisions must be detected and

solved without causing deadlocks. Navigation is typically solved using a pathfinding algorithm, of which A* is the most common. Extensive work has been made in optimizing A* to improve the performance of the pathfinding in video games. In Higgins (2002) some tricks that were used to optimize the pathfinding engine in the RTS game *Empire Earth* are mentioned. Demyen and Buro (2008) address the problem of abstracting only the information from the gameworld that is useful for the pathfinder engine by using triangulation techniques. In Koenig and Likhachev (2006) an approach for improving the performance of A* in adaptive game worlds by updating heuristics in nodes based on previous searches is described. Additional work on adaptive A* can be found in Sun, Koenig, and Yeoh (2008) where the authors propose a Generalized Adaptive A* method that improve perfomance in game worlds where the action cost for moving from one node to another can increase or decrease over time.

In RTS games bots often have complete visibility of the game world in contrast to the limited view a human player has, i.e. the bots "cheat". The purpose is to have as much information as possible available to the AI to reason about how to approach the enemy, find tactically good positions such as choke points, and not spending time exploring the game world to locate resources. Cheating is, according to Nareyek (2004), *"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 game AIs. Human players usually only have complete visibility of areas around own units and bases, rest of the game world is unknown. This is usually referred to as *Fog of War* or FoW.

1.2 Artificial Potential Fields

Artificial Potential Fields is a concept originating from robotics. It was first introduced by Khatib (1986) for real-time obstacle avoidance for manipulators and mobile robots. The technique works by placing manipulators in a field of forces. The position to be reached by the robot is an attracting manipulator, while obstacles are repelling manipulators. Many studies concerning potential fields are related to spatial navigation and obstacle avoidance, for example the work by Borenstein and Koren (1991) and Massari, Giardini, and Bernelli-Zazzera (2004). Potential fields has also been used for obstacle avoidance in several video games. Alexander (2006) describes the use of flow fields, which has similarities with potential fields, for obstacle avoidance in the games *Blood Wake* and *NHL Rivals*. Johnson (2006) described obstacle avoidance using repulsion vectors in the game *The Thing*.

With the exception of obstacle avoidance combined with pathfinding, potential fields has had limited success in games. Thurau, Bauckhage, and Sagerer (2004b) have developed a bot for the first-person shooter game *Quake II* based on potential fields. The bot learns reactive behaviors by observing human players and, based on its observations, it adapts weights of potential fields to learn tactically good positions and paths in the

game world. Wirth and Gallagher (2008) used potential fields in the game *Ms.Pacman*. Potential fields has also been used in robot soccer (Johansson & Saffiotti, 2002; Röfer et al., 2004).

To give a brief introduction to how potential fields can be used in RTS games a simple resource gathering scenario will be described. This is a one-player scenario where the player has one base and a number of worker units. The aim is to move workers to a mine, gather as much resources as each worker can carry, then return to the base to drop them off. The workers must also avoid colliding with terrain, dynamic objects such as other own workers, and the own base.

Each worker has two states; *Mine* and *DropOff*. In the *Mine* state the driving force for the workers is to move to nearby mines, and therefore an attracting charge is placed at the position of each mine. These charges are spread in the game world and gradually fades to zero. Terrain, own worker units and the base all generates small repelling fields used for obstacle avoidance. Figure 1.1 illustrates a miner (white circle) moving from the base to a nearby mine. An attractive charge that generates a large field is placed at the center of the mine. Lighter areas in the field are more attracting than darker areas. The figure also shows the small repelling fields (slightly darker grey) around the worker, terrain and the base.

Figure 1.2 illustrates a worker in the *DropOff* state. The worker now carries as much resources as it can, and must move to the base to drop them off. The attractive charge is now placed in the center of the base, and the mine instead of the base generates a small repelling field for obstacle avoidance.

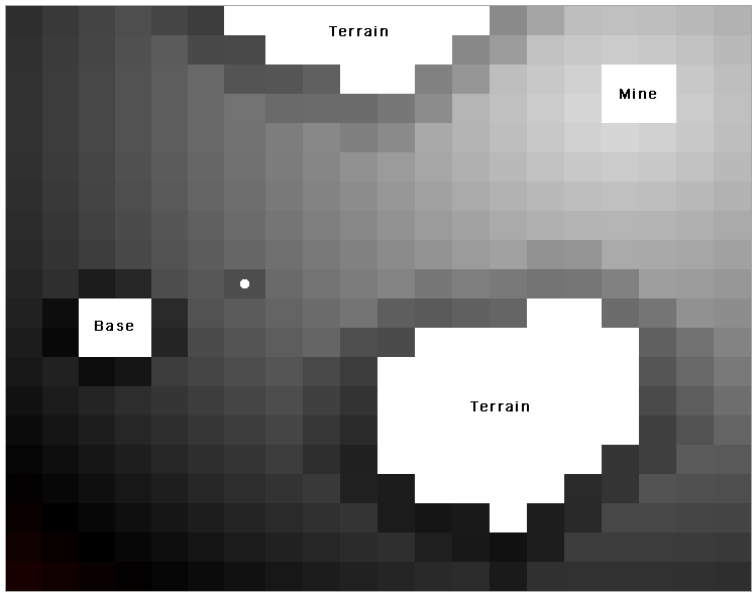

Figure 1.1: *A worker unit (white circle) moving towards a mine to gather resources. The mine generates an attractive field used by the worker for navigation. Mountains (black) generate small repelling fields for obstacle avoidance. Light grey areas are more attracting than darker grey areas.*

The use of potential fields in games has been limited. There are a number of more or less good arguments for that:

- Agents navigating using PFs may get stuck in local optima. This can for example happen when the path to the destination is blocked by a large mountain.
- PF based solutions are believed to require more memory and CPU resources than traditional A* based solutions.
- PF based solutions are believed to be hard to implement, tune and difficult to debug.
- PFs are considered to be less controllable than traditional solutions.

In this thesis we show that these issues can be more or less solved and that potential fields can be used for navigating units in a RTS scenario.

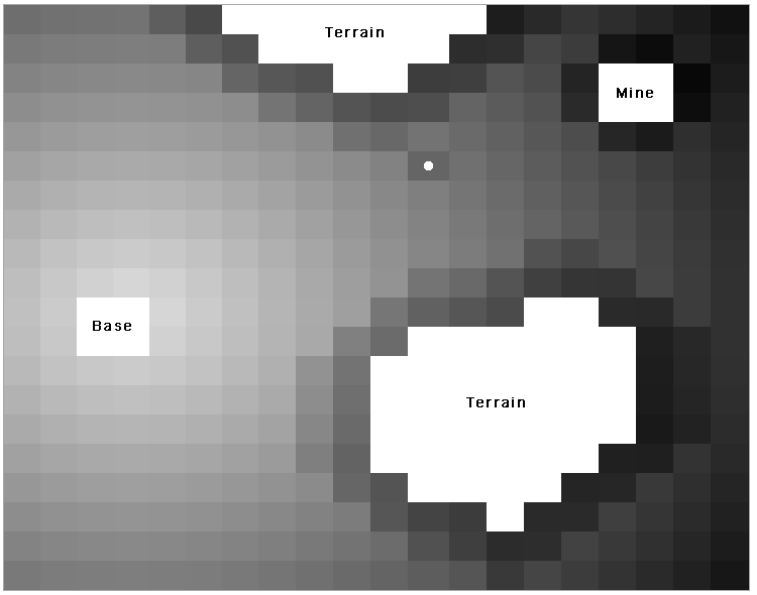

Figure 1.2: *A worker unit (white circle) moving towards a base to drop of gathered resources. Now the base generates an attractive field used by the worker for navigation.*

1.3 Research Questions

The main goal of this thesis is to evaluate if multi-agent potential fields (MAPF) is a viable option for controlling armies in different real-time strategy game scenarios. It involves performance in terms of being able to defeat its opponents, performance in terms of computational resources used, ability to adapt to different scenarios and ability to handle incomplete information about the game world. The three following research questions are addressed:

RQ1. How does a MAPF based solution perform compared to traditional solutions?

This question is answered by studying performance in terms of playing specific RTS scenarios against other opponents. It involves both performance in terms of playing the game well and defeat the opponents, and performance in terms of computational resources used.

RQ2. To what degree is MAPF an approach that is modifiable with respect to variations in the game scenario?

RTS games can be very complex and can be varied endlessly, for example different terrain, different types and combination of units, different scenarios, different conditions for winning a game and much more. This question is answered by studying how modifiable a MAPF based approach is regarding changes in a scenario and its ability to be adapted for different scenarios and goals.

RQ3. To what degree is a MAPF based solution able to handle incomplete information about the game world?

Computer players in RTS games often cheat in the sense that they have complete visibility and perfect information about the whole game world, while human players only have complete visibility of areas surrounding own units and buildings. We study if a MAPF based approach is viable in a scenario with incomplete information, i.e. fog of war, and we measure its performance against bots with complete information about the game world.

1.4 Research Methods

The research questions have been answered using a quantitative approach. We have implemented a Multi-Agent Potenial Field based bot for the open-source ORTS engine. The bot has been tested in the yearly ORTS competition organized by the University of Alberta. As a testbed, we believe that the tournament 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. In addition to tournaments, we have conducted experiments in a controlled and designed environment. In the experiments we have used the top teams from the 2007 years' tournament as opponents. We believe that using opponent teams from the tournament in the experiments is more trustworthy compared to using bots developed by ourselves as opponents.

1.5 Contributions

In this section, we address the contribution and give a brief summary of each paper.

1.5.1 RQ1: How does a MAPF based solution perform compared to traditional solutions?

RQ1 is addressed in Papers I, II and III. In Paper I we present a methodology containing six phases for designing multi-agent potential fields (MAPF) based bots for real-time strategy games. A bot for the open RTS game engine ORTS was developed and is evaluated in the Tankbattle scenario of 2007 years' version of an annual open ORTS tournament organised by the University of Alberta. Even though the bot performed poorly and was only able to win 32% of the played games, the approach showed some promises. In Paper III some issues and weaknesses of the bot described in Paper I were identified and addressed. The MAPF based bot was improved, and in the experiments that we conducted was able to beat the four top teams from 2007 years' ORTS tournament with an almost perfect score. In addition the bot was evaluated in 2008 years' ORTS tournament where it won the Tankbattle scenario with 98% wins of the games played. Paper II is a demo paper describing a demonstration of the improved bot from Paper III. Our conclusion is that MAPF based bots is a viable approach in some RTS scenarios being able to match and surpass the performance of more traditional solutions.

1.5.2 RQ2: To what degree is MAPF an approach that is modifiable with respect to variations in the game scenario?

RQ2 is addressed in Paper V where we show how a MAPF based bot can be modified to handle more complex scenarios such as a Full RTS game. In addition to the bot described in Papers I-IV, Paper V describes how to deal with resource gathering, base building and high-level tactics. The bot was evaluated in 2008 years' ORTS tournament where it won the Full RTS scenario with a win percent of 82.5% of the games played. Our conclusion is that a MAPF based bot can easily be modified to changes in the setup and goal of a scenario and still match the perfomance of traditional solutions, thus answering RQ2.

1.5.3 RQ3: To what degree is a MAPF based solution able to handle incomplete information about the game world?

RQ3 is addressed in Paper IV where we show how a MAPF based bot designed for scenarios with complete information can be modified to handle incomplete information of the game world, i.e. fog of war (FoW). We conclude that a bot without complete information can perform equally good or even surpass a bot with complete information without using more computational resources.

1.6 Discussion and Conclusions

First we list a number of criticisms against potential field based solutions with our point of view:

- *PFs have issues like local optima that are difficult to solve.* With the use of pheromone trails as described in Paper II, many local optima issues can be solved. PFs still have problems in very complex terrain such as mazes, and in those cases pathfinding based methods are probably better suited. The strength of PFs are in large dynamic worlds with large open areas and less complicated terrain, and its ability to handle multiple objectives for the agents. This is the case for many RTS games of today. There is a need to further compare PF's with A*, both in terms of ability to navigate and computational resources used, in a number of fixed scenarios.

- *PF based solutions use too much resources.* In Paper II and Paper IV we show that PF based solutions can be implemented with the same or better resource efficiency as pathfinding based methods. More experiments carried out in other environments have to be done to back up this statement. Efficiency can still be a problem, but that can be the case in path planning based systems as well.

- *PF based solutions are difficult to implement and tune.* PFs can be implemented with very simple architectures. Tuning can be difficult and time consuming, but the relative importance between the fields is a great help here (i.e. is destroying a base more important than destroying units?). A graphical representation of the potential field view is also valuable. There is however a need for better tools and methodologies to aid in the calibration process.

- *PFs are considered to be less controllable than traditional solutions.* In a PF based solution an often large amount of fields collaborate to form the total potential field which is used by the agents for navigation in the game world. This can lead to interesting emergent behaviors, but can also limit the control over agents compared to path planning solutions. A debug tool with graphical representation of the potential fields is of great value in this matter.

We believe that potential field based solutions can be a successful option to more conventional path planning based solutions in many RTS scenarios. Papers I, II and III show that the method is able to match and even surpass the performance of state-of-the-art solutions. The work with incomplete information also shows that PF based solutions are highly configurable and can adapt well to changes in the game scenario. Another benefit of using potential fields is the use of non-monotonous fields which can lead to interesting emergent behavior. In our bot we used a field with the most attractive potential at radius *maximum shooting distance* from enemy tanks, and in combination with the small

repelling field around own units lead to our forces being able to surround the enemy at an appropriate distance. We also show that by simply changing the shape and weights of potential fields, agents can adapt to changes of the goal. For example an agent controlling a tank move towards enemy units when its weapon is ready to fire, and retreat outside firerange when the weapon needs to be reloaded. In addition we show how enemy units and bases can generate different fields for different unit types to, for example, reflect the shorter *maximum shooting distance* of marines compared to tanks.

We also believe that potential field based solutions can be a successful option in other game genres such as sports and First Person Shooters (FPS). In many sport games, for example soccer and basketball, agents have to navigate in open areas without static obstacles but with lots of dynamic obstacle avoidance which suit potential fields well. Thurau et al. (2004b) describes how potential fields can be used to imitate human movement behavior in the FPS game *Quake II*.

1.7 Future Work

We define four possible directions for future work; validation in other domains, 3D worlds, technical refinements, and adaptivity and player experience.

1.7.1 Validation in Other Domains

Our multi-agent potential field methodology has been validated in the ORTS engine. We have also done some initial testing in the Warcraft II clone Wargus, but it is too early to draw any conclusions about the performance in that game engine. There is a need to furher test a MAPF based solution in Wargus, and to test and validate the methodology in other RTS platforms.

1.7.2 3D worlds

The ORTS engine uses the, for RTS games, very common top-down view where the player gets a 3D feeling of the game, but the low-level navigation system is running in 2D. A direction for future work would be to try a potential field based navigation system in a full 3D game where units have an (x, y, z) position in the game world. Full 3D is rarely used in RTS games and the FPS genre is probably more suited for this direction of future work. One major problem that needs to be addressed is that navigation is typically solved by projecting a 2D graph onto the 3D world, and in such a graph Euclidean distance cannot be used.

1.7.3 Technical Refinements

This direction involves investigating ideas about how to improve performance or behavior of potential field based solutions. Possible improvements are:

- *Non-symmetric directional fields.* In our bot the fields generated around enemy units are circular and symmetric. If, for example, the enemy has stationary machine gun units, the generated fields can be repelling in the shooting direction of the machine gun, and attracting in the flanks and the back of it. This will lead to a situation where own units avoid the line-of-fire of the machine gun and try to attack it from the flanks or the back. This can also involve investigating the cost of using non-symmetric fields in terms of increased memory and/or CPU usage.

- *Use the GPU to improve performance.* Potential field generation require a lot of float value calculations. Graphics processors, GPUs, are optimized for float operations and has been used with success for improving performance in many scientific problems (Owens et al., 2008). The idea is to improve computational performance by using the GPU for field generation.

- *Combinations of pathfinding and potential fields.* In some situations it might be beneficial to combine pathfinding with potential fields in order to benefit from both techniques. Units can for example use pathfinding for navigation over longer distances, and switch to potential fields in combat situations which often involves a large number of dynamic objects.

- *Tactical behaviors.* Some positions in the game world has a high tactial importance. This can for example include choke points where the enemy forces can be ambushed, and areas with lots of resources. In our work potential fields has mostly been generated by physical objects in the game world, but the idea is to let tactically important positions generate fields as well.

- *Caching of PF values.* In our implementation the potential field value in a certain point is recalculated every time that point is evaluated by a agent, even if the potential in the point have previously been calculated. A caching of calculated potential field values can probably improve performance of the bot. When caching values it is important to take in consideration that different subfields can be active or inactive depending on the internal state of each agent, and therefore the potential field value in a point can be different between the agents.

- *Use machine learning and optimization techniques to tune potential fields.* Tuning of potential fields can be difficult and time consuming. The idea is to automate tuning by using techniques such as genetic algorithms, neural networks etc. We identify two things that may be an issue for this approach: i) evaluation can be time consuming if full games must be played against one or maybe more opponents,

and ii) there may be a lot of random factors in a game (for example the damage a unit make in an attack) that has impact on convergence. Still the idea is interesting.

1.7.4 Adaptivity and Player Experience

This direction is about how potential fields can be adapted to suit player skills. This includes difficulty scaling at runtime to make games even where neither the computer player nor the human player is outstanding. It also includes change in tactics at runtime to match the player, for example switching between defensive and attacking behavior depending on how the human player controls his army. An important aspect here is to investigate how adaptivity and difficulty scaling affect player experience. With automated experiments we can show that the bot adapts and that the difficulty level is balanced, but is it more fun for the human player than a non-adaptive solution?

CHAPTER

TWO

PAPER I

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

Johan Hagelbäck & Stefan J. Johansson
Proceedings of the Seventh International Conference on Autonomous Agents and Multi-agent Systems (AAMAS). 2008.

2.1 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.

Khatib (1986) 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. Later on 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.

Many studies concerning potential fields are related to spatial navigation and obstacle avoidance, see e.g. Borenstein and Koren (1991); Khatib (2004); Massari et al. (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). 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 (Vadakkepat, Tan, & Ming-Liang, 2000).

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, Matarić, and Sukhatme (2002) developed a mobile sensor network deployment using potential fields, and potential fields have been used in robot soccer (Johansson & Saffiotti, 2002; Röfer et al., 2004). Thurau et al. (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.

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 (2003) and early work by vanLent et al. (1999). Multi-agent systems has been used in board games by Kraus and Lehmann (1995) who addressed the use of MAS in Diplomacy and Johansson (2006) who proposed a general MAS architecture for board games.

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 2.3 we describe the test environment. The creation of our MAPF player follows the proposed methodology and we report on that in Section 2.4. The experiments and their results are described in Section 2.5. We finish off by discussing, drawing some conclusions and outlining future work in Sections 2.6–2.7.

2.2 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).

2.3 ORTS

Open Real Time Strategy (ORTS) (Buro, 2007a) 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 (Buro, 2007a).

- 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.

2.4 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.

2.4.1 Identifying objects

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

2.4.2 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. 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.

2.4.3 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.

Figure 3.2 in Paper II shows a 2D view of the map during a tankbattle game. It shows our agents (green) moving in to attack enemy bases and units (red). Figure 3.3 shows the potential field view of the same tankbattle game. Dark areas has low potential and light areas high potential. The light ring around enemy bases and units, located at maximum

shooting distance of our tanks, is the distance our agents prefer to attack opponent units from. It is the final move goal for our units.

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

$$p_{cliff}(d) = \begin{cases} -80/d^2 & \text{if } d > 0 \\ -80 & \text{if } d = 0 \end{cases} \tag{2.1}$$

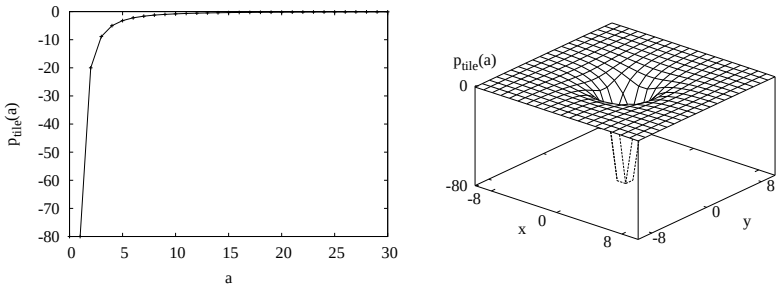

Figure 2.1: *The potential $p_{cliff}(d)$ generated by a cliff given the distance d.*

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 2.1.

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 in Figure 2.2 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
```

Figure 2.2: *Pseudo code for filling dead ends.*

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 2.3).

Figure 2.3: *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.*

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 2.5. Figure 2.4 shows the pseudo code for this processing step. 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.

```
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
```

Figure 2.4: *Pseudo code for clearing narrow passages.*

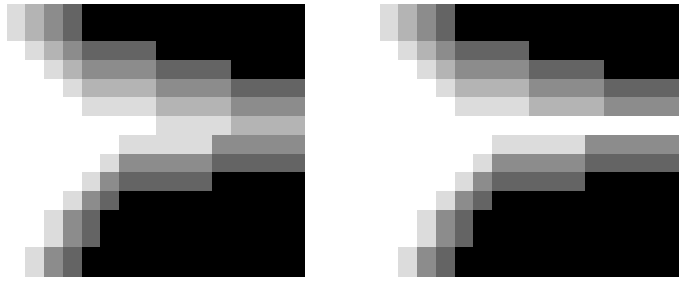

Figure 2.5: *Example of the navigation field before and after clearing passages. White tiles has potential 0, and the darker the colour the more negative potential a tile has.*

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 2.6, 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:

$$p(d) = \begin{cases} k_1 d, & \text{if } a \in [0, MSD - a[\\ c_1 - d, & \text{if } a \in [MSD - a, MSD] \\ c_2 - k_2 d, & \text{if } a \in]MSD, MDR] \end{cases} \tag{2.2}$$

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 2.1: *The parameters used for the generic $p(d)$-function of Eq. 2.2.*

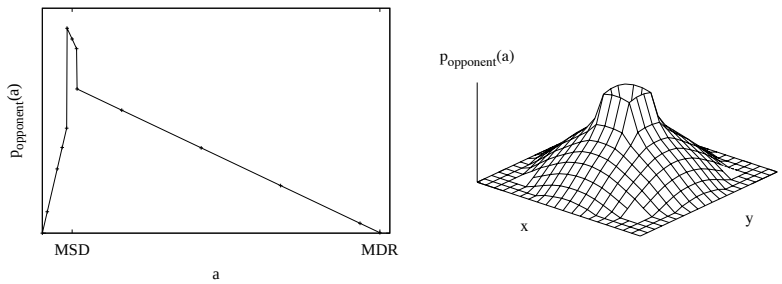

Figure 2.6: *The potential $p_{opponent}(d)$ generated by the general opponent function given the distance d.*

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 circular, see Figure 2.7), but it works well in practice.

$$p_{ownbase}(d) = \begin{cases} 5.25 \cdot d - 37.5 & \text{if } d <= 4 \\ 3.5 \cdot d - 25 & \text{if } d \in]4, 7.14] \\ 0 & \text{if } d > 7.14 \end{cases} \quad (2.3)$$

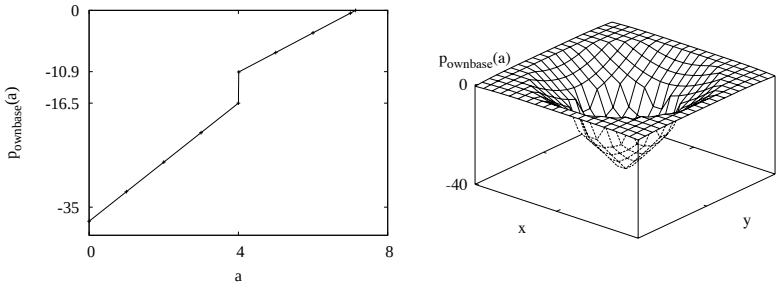

Figure 2.7: *The repelling potential $p_{ownbase}(d)$ generated by the own bases given the distance d.*

The own mobile units — tanks and marines Own units, agents, generate a repelling field for obstacle avoidance (see Figure 2.8). In general terms, the potential $p_{ownunit}(d)$ at distance d (in tiles) from the center of an agent is calculated as:

$$p_{ownunit}(d) = \begin{cases} -20 & \text{if } d <= radius \\ d \cdot k - c & \text{if } d \in]radius, l], \\ 0 & \text{if } d >= l \end{cases} \tag{2.4}$$

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

Table 2.2: *The parameters used for the generic $p_{ownunit}(d)$-function of Eq. 2.4.*

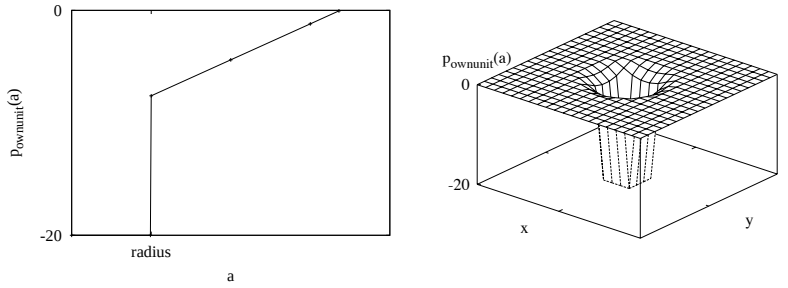

Figure 2.8: *The repelling potential $p_{ownunit}(d)$ generated by the generic function given the distance d.*

Sheep Sheep generate a small repelling field for obstacle avoidance. The potential $p_{sheep}(d)$ (depicted in Figure 2.9) at distance d (in tiles) from the center of a sheep is calculated as:

$$p_{sheep}(d) = \begin{cases} -10 & \text{if } d <= 1 \\ -1 & \text{if } d \in]1, 2] \\ 0 & \text{if } d > 2 \end{cases} \tag{2.5}$$

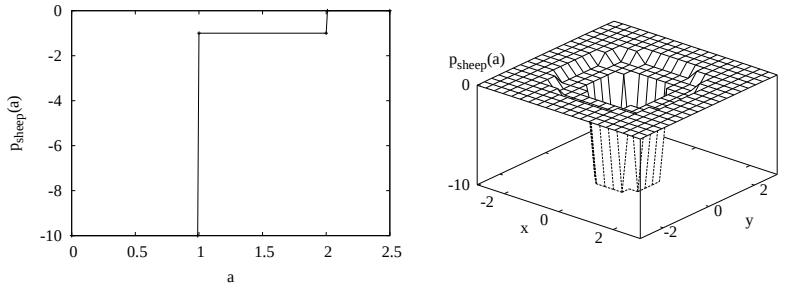

Figure 2.9: *The potential $p_{sheep}(d)$ generated by a sheep given the distance d.*

2.4.4 On the granularity

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.

2.4.5 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.

2.4.6 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:

- 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.

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

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:

$$a_{k,i} = \begin{cases} 1 & \text{if the agent } k \text{ can attack opponent unit } i \\ 0 & \text{if the agent } k \text{ cannot attack opponent unit } i \end{cases} \tag{2.6}$$

$$A = \begin{bmatrix} a_{0,0} & \cdots & a_{m-1,0} \\ \vdots & \ddots & \vdots \\ a_{0,n-1} & \cdots & a_{m-1,n-1} \end{bmatrix} \tag{2.7}$$

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

$$HP = \begin{bmatrix} HP_0 \\ \vdots \\ HP_{n-1} \end{bmatrix} \tag{2.8}$$

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

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad HP = \begin{bmatrix} [h]HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_3 = 4 \\ HP_4 = 4 \\ HP_5 = 3 \end{bmatrix} \tag{2.9}$$

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:

$$A_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad HP = \begin{bmatrix} [h]HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_5 = 3 \\ HP_4 = 4 \\ HP_3 = 4 \end{bmatrix} \tag{2.10}$$

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.

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & & & 0 & \\ 0 & 0 & 0 & 0 & & & 0 & \\ 0 & 0 & 0 & 0 & & & 0 & \\ 0 & 0 & 0 & 0 & & & 0 & \end{bmatrix} \quad HP = \begin{bmatrix} HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_5 = 3 \\ HP_4 = 4 \\ HP_3 = 4 \end{bmatrix} \tag{2.11}$$

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:

$$A_4 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad HP = \begin{bmatrix} HP_0 = 2 \\ HP_1 = 3 \\ HP_2 = 3 \\ HP_5 = 3 \\ HP_4 = 4 \\ HP_3 = 4 \end{bmatrix} \tag{2.12}$$

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.

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:

$$(\sum_{k=0}^{m-1} a(k,i)) \cdot (damage_u - armour_i) >= HP_i \tag{2.13}$$

$armour_i$ 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.

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

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.

2.5 Experiments

Our bot have participated in the 2007 years ORTS competition. Below is a brief description of the other competition entries (Buro, 2007a). The results from the competition are presented in Tables 2.3–2.4. 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.

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 2.3: *Summary of the results of ORTS tank-battle 2007*

2.5.1 Opponent Descriptions

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

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 2.4: *Summary of the results of the ORTS tactical combat*

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.

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.

2.6 Discussion

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

2.6.1 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 (Tomlinson, 2004). 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 (Tomlinson, 2004).

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. (2004b) 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.

2.6.2 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 increase 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 (2004)).

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.

2.6.3 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.

2.7 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 2.6, 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.

CHAPTER

THREE

PAPER II

Demonstration of Multi-agent Potential Fields in Real-time Strategy Games

Johan Hagelbäck & Stefan J. Johansson
Demo Paper on the Seventh International Conference on Autonomous Agents and Multi-agent Systems (AAMAS). 2008.

3.1 The ORTS environment

Open Real Time Strategy (ORTS) (Buro, 2007a) is a real-time strategy game engine developed as a tool for researchers within artificial intelligence (AI) in general and game AI in particular, see Figure 3.1. 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.

Figure 3.1: *The 3D view of the ORTS Tankbattle game.*

3.2 The used technology

Khatib (1986) 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.

Although being a well-known technology in robotics, potential fields has not gained very much interest in the game industry. We show that, not only is it an efficient and robust solution for navigation of a single unit, it is also an approach that works very well in distributed settings of multiple agents. Figure 3.3 shows the potential fields for the green team.

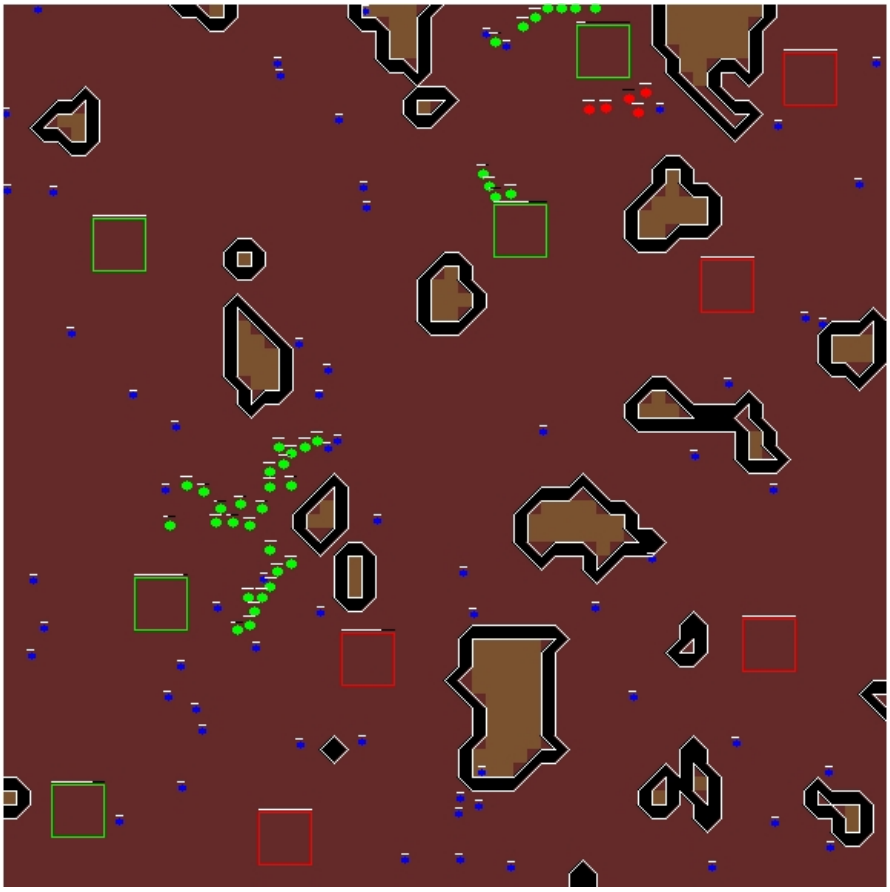

Figure 3.2: *The 2D view of the same ORTS Tankbattle game.*

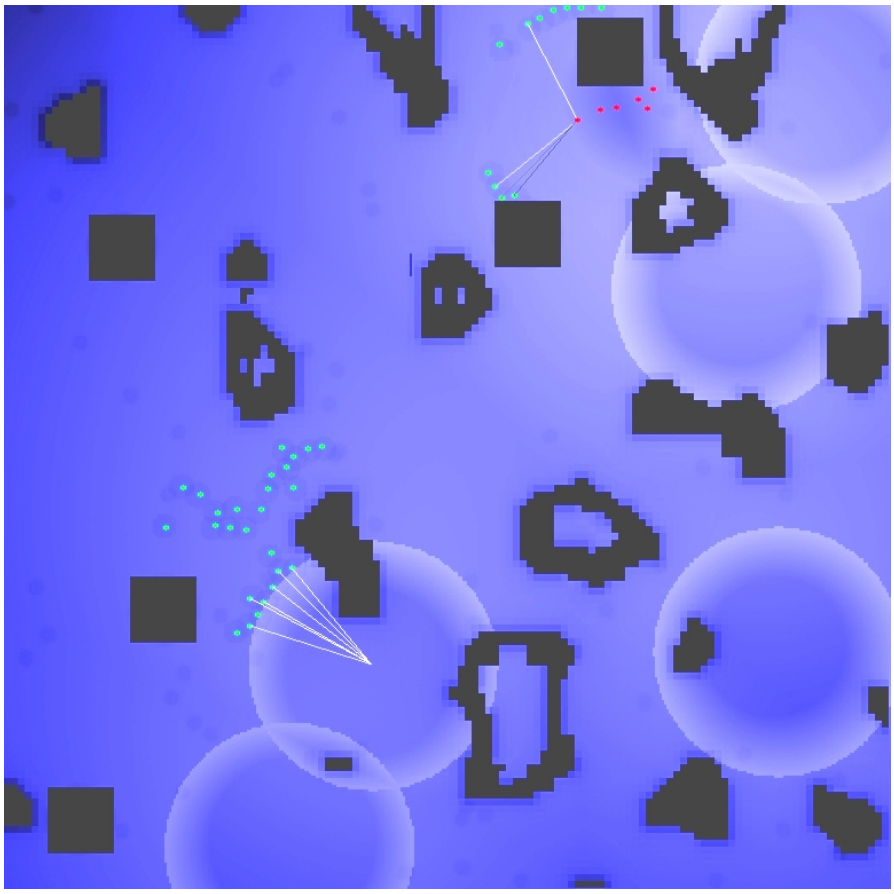

Figure 3.3: *The potential field generated by the units and the terrain. The white lines illustrate the coordinated attacks on a base (lower left) and a unit (upper right).*

3.3 The involved multi-agent techniques

There are several issues to be addressed in an RTS game. First, all units are moving in parallel, which means that they will have to coordinate their movement in some way without bumping into each other, or the surrounding environment. We use potential fields similar to the ones used by e.g. Mamei and Zambonelli (2004) to let the units keep themselves at the right distance.

Second, to improve the efficiency, we coordinate their attacks through the use of a central military commander. This agent is not embodied in the field, but makes sure that no extra shots are spent on opponent units that are already under lethal attack. This is important, since there is a cool-down period during which the units can not attack after a shot.

Third, the commander chooses what opponent to attack first. This is a strategic decision that may follow several strategies, e.g. to try to split the enemies in more, but weaker groups, or try to attack the enemy from the sides. In order to make the right decision, an analysis of the spatial (and health-related) positions of the opponent agents is needed.

3.4 The innovation of the system

The use of *separate potential fields* for the control of tactical, navigational, and strategic matters in a system of multiple units (our agents) in an RTS game has, as far as we know, not been described in academia before. Traditionally, A* and different types of state machines has been state-of-the-art in the gaming industry. Lately we have seen a growing interest for alternative solutions, partly as a result of the customer demand for more believable computer opponents, partly as a result of the increase in processing power that third generation game consoles such as Sony PlayStation 3 bring us. We believe that the use of both MAS techniques and potential fields (and why not our proposed combination of the two?) will gain ground as the game AI field matures. Lately, the performance of our solution has increased significantly compared to the results presented in Paper I and these late breaking improvements will of course be demonstrated.

3.5 The interactive aspects

Unfortunately, the human player interface is not yet released by the ORTS developers. If it will be available at the time of the conference, we will also be able to offer the audience to play games against our MAPF based bot. If not, we will illustrate its features through games against other computer opponents. We will be glad to illustrate the performance of our recently updated solution against the winner of the ORTS tournament described in Paper I.

There will be two windows updated in real time. The main window shows a 3D (see Figure 3.1), or 2D (see Figure 3.2) view of the units and the terrain. The second window (see Figure 3.3) shows the potential fields of a certain unit, as well as the resulting coordination done by the military commander. The whole potential field is shown here, although in the real application, only the potentials of the positions in the map that are considered interesting are calculated.

3.6 Conclusions

We will show a demonstration of a highly competitive game AI bot for the ORTS environment. It is built using the methodology described in Paper I and use a combination of Multi-agent coordination techniques and potential fields to try to win its games.

CHAPTER

FOUR

PAPER III

The Rise of Potential Fields in Real Time Strategy Bots

Johan Hagelbäck & Stefan J. Johansson
Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE). 2008.

4.1 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 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.

Khatib (1986) 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. Later on 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.

Many studies concerning potential fields are related to spatial navigation and obstacle avoidance, see e.g. Borenstein and Koren (1991); Massari et al. (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. (2002) developed a mobile sensor network deployment using potential fields, and potential fields have been used in robot soccer (Johansson & Saffiotti, 2002; Röfer et al., 2004). Thurau et al. (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. In Paper I we have 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.

4.2 ORTS

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

4.2.1 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 4.1). 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:

$$\begin{aligned} score = & 5 \cdot (ownBasesLeft - oppBasesLeft) + \\ & ownUnitsLeft - oppUnitsLeft \end{aligned} \tag{4.1}$$

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

4.2.2 Opponent descriptions

We refer to Paper I section 2.5.1 for opponent descriptions.

4.3 MAPF in ORTS, V.1

We have implemented an ORTS client for playing Tankbattle based on Multi-agent Potential Fields (MAPF) following the proposed methodology in Paper I. 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.

4.3.1 Identifying objects

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

4.3.2 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 precalculated 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.

4.3.3 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]

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

The opponent units

$$p(d) = \begin{cases} k_1 d, & \text{if } d \in [0, MSD - a[\\ c_1 - d, & \text{if } d \in [MSD - a, MSD] \\ c_2 - k_2 d, & \text{if } d \in]MSD, MDR] \end{cases} \tag{4.2}$$

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 4.2: *The parameters used for the generic $p(d)$-function of Equation 4.2.*

Own bases Own bases generate a repelling field for obstacle avoidance. Below in Equation 4.3 is the function for calculating the potential $p_{ownB}(d)$ at distance d (in tiles) from the center of the base.

$$p_{ownB}(d) = \begin{cases} 5.25 \cdot d - 37.5 & \text{if } d <= 4 \\ 3.5 \cdot d - 25 & \text{if } d \in]4, 7.14] \\ 0 & \text{if } d > 7.14 \end{cases} \tag{4.3}$$

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

$$p_{ownU}(d) = \begin{cases} -20 & \text{if } d <= 0.875 \\ 3.2d - 10.8 & \text{if } d \in]0.875, l], \\ 0 & \text{if } d >= l \end{cases} \tag{4.4}$$

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:

$$p_{sheep}(d) = \begin{cases} -10 & \text{if } d <= 1 \\ -1 & \text{if } d \in]1, 2] \\ 0 & \text{if } d > 2 \end{cases} \tag{4.5}$$

Figure 3.2 in Paper II shows a 2D view of the map during a tankbattle game. It shows our agents (green) moving in to attack enemy bases and units (red). Figure 3.3 shows the

potential field view of the same tankbattle game. Dark areas has low potential and light areas high potential. The light ring around enemy bases and units, located at maximum shooting distance of our tanks, is the distance our agents prefer to attack opponent units from. It is the final move goal for our units.

4.3.4 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.

4.3.5 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 Paper I.

4.4 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.

4. Our units came too close to the nearest opponents if the opponent units were gathered in large groups.

We will now describe four different ways to address the identified problems by adjusting the original bot V.1 described in Paper I. The modifications are listed in Table 4.3.

Properties	V.1	V.2	V.3	V.4	V.5
Full resolution		√	√	√	√
Defensive field			√	√	√
Charged pheromones				√	√
Max. potential strategy					√

Table 4.3: *The implemented properties in the different experiments using version 1–5 of the bot.*

4.4.1 Increasing the granularity, V.2

In the original ORTS bot we used 128x128 tiles for the potential field, where each tile was 8x8 positions in the game world. The potential field generated from a game object, for example own tanks, was pre-calculated in 2-dimensional arrays and simple copied at runtime into the total potential field. This resolution proved not to be detailed enough. In the tournament our units often got stuck in terrain or other obstacles such as our own bases. This became a problem, since isolated units are easy targets for groups of attacking units.

The proposed solution is to increase the resolution to 1x1 positions per tile. To reduce the memory requirements we do not pre-calculate the game object potential fields, instead the potentials are calculated at runtime by passing the distance between an own unit and each object to a mathematical formula. To reduce computation time we only calculate the potentials in the positions around each own unit, not the whole total potential field as in the original bot. Note that the static terrain is still pre-calculated and constructed using 8x8 positions tiles. Below is a description and formulas for each of the fields. In the experiments we use weight $1/7 \approx 0.1429$ for each of the weights w_1 to w_7. The weight w_7 is used to weight the terrain field which, except for the weight, is identical to the terrain field used in the original bot. The results from the experiments are presented in Table 4.4. Below is a detailed description of the fields.

Team	Win %	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 4.4: *Experiment results from increasing the granularity.*

The opponent units and bases. All opponent units and bases generate symmetric surrounding fields where the highest potentials surround the objects at radius D, the

MSD, R refers to the *Maximum Detection Range*, the distance from which an agent starts to detect the opponent unit. The potentials $p_{oppU}(d)$ and $p_{oppB}(d)$ at distance d from the center of an agent are calculated as:

$$p_{oppU}(d) = w_1 \cdot \begin{cases} 240/d(D-2), & \text{if } d \in [0, D-2[\\ 240, & \text{if } d \in [D-2, D] \\ 240 - 0.24(d-D) & \text{if } d \in]D, R] \end{cases} \tag{4.6}$$

$$p_{oppB}(d) = w_6 \cdot \begin{cases} 360/(D-2) \cdot d, & \text{if } d \in [0, D-2[\\ 360, & \text{if } d \in [D-2, D] \\ 360 - (d-D) \cdot 0.32 & \text{if } d \in]D, R] \end{cases} \tag{4.7}$$

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

$$p_{ownU}(d) = w_3 \cdot \begin{cases} -20 & \text{if } d <= 14 \\ 32 - 2 \cdot d & \text{if } d \in]14, 16] \end{cases} \tag{4.8}$$

Own bases. Own bases also generate repelling fields for obstacle avoidance. Below is the function for calculating the potential $p_{ownB}(d)$ at distance d from the center of the base.

$$p_{ownB}(d) = w_4 \cdot \begin{cases} 6 \cdot d - 258 & \text{if } d <= 43 \\ 0 & \text{if } d > 43 \end{cases} \tag{4.9}$$

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

$$p_{sheep}(d) = w_5 \cdot \begin{cases} -20 & \text{if } d <= 8 \\ 2 \cdot d - 25 & \text{if } d \in]8, 12.5] \end{cases} \tag{4.10}$$

4.4.2 Adding a defensive potential field, V.3

After a unit has fired its weapon the unit has a cooldown period when it cannot attack. In the original bot our agents was, as long as there were enemies within MSD (D), stationary until they were ready to fire again. The cooldown period can instead be used for something more useful and we propose 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 D of the closest opponent unit or base and surround the opponent unit cluster at D. The potential $p_{def}(d)$ at distance d from the center of an agent is calculated using the formula in Equation 4.11. The results from the experiments are presented in Table 4.5.

$$p_{def}(d) = w_2 \cdot \begin{cases} w_2 \cdot (-800 + 6.4 \cdot d) & \text{if } d <= 125 \\ 0 & \text{if } d > 125 \end{cases} \tag{4.11}$$

Team	Win %	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

Table 4.5: *Experiment results from adding a defensive field.*

4.4.3 Adding charged pheromones, V.4

The local optima problem is well known in general when using PF. Local optima are positions in the potential field that has higher potential than all its neighbouring positions. A unit positioned at a local optimum will therefore get stuck even if the position is not the final destination for the unit. In the original bot agents that had been idle for some time moved in a random direction for some frames. This is not a very reliable solution to the local optima problem since there is not guarantee that the agent has moved out of, or will not directly return to, the local optima.

Thurau, Bauckhage, and Sagerer (2004a) described 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 optima.

We have introduced a trail that adds a negative potential to the last 20 positions of each agent. Note that an agent is not effected by the trails of other own agents. The negative potential for the trail was set to -0.5 and the results from the experiments are presented in Table 4.6.

Team	Win %	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 4.6: *Experiment results from adding charged pheromones.*

4.4.4 Using maximum potentials, V.5

In the original bot all potential fields generated from 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 centre of the opponent unit cluster. This makes our agents attack the centre of the enemy force 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 moving towards the centre of the opponent unit cluster. The results from the experiments are presented in Table 4.7.

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 4.7: *Experiment results from using maximum potential, instead of summing the potentials.*

4.5 Discussion

The results clearly show that the improvements we suggest increases the performance of our solution dramatically. We will now discuss these improvements from a wider perspective, asking ourselves if it would be easy to achieve the same results without using potential fields.

4.5.1 Using full resolution

We believed that the PF based solution would suffer from being slow. Because of that, we did not initially use the full resolution of the map. However, we do so now, and by only calculating the potentials in a number of move candidates for each unit (rather than all positions of the map), we have no problems at all to let the units move in full resolution. This also solved our problems with units getting stuck at various objects and having problems to go through narrow passages.

4.5.2 Avoiding the obstacles

The problems with local optima are well documented for potential fields. It is a result of the lack of planning. Instead, a one step look-ahead is used in a reactive manner. This is of course problematic in the sense that the unit is not equipped to plan its way out of a sub-optimal position. It will have to rely on other mechanisms. The pheromone trail is one such solution that we successfully applied to avoid the problem.

On the other hand, there are also advantages of avoiding to plan, especially in a dynamically changing environment where long term planning is hard.

4.5.3 Avoiding opponent fire

The trick to avoid opponent fire by adding a defensive potential field during the cool-down phase is not hard to implement in a traditional solution. By adding a state of cool-down, which implements a flee behaviour, that makes the unit run away from the

enemies, that could be achieved. The potential problem here is that it may be hard to coordinate such a movement with other units trying to get to the front, so some sort of coordinating mechanism may be needed. While this mechanism is implicit in the PF case (through the use of small repulsive forces between the own units), it will have to be taken care of explicitly in the planning case.

4.5.4 Staying at maximum shooting distance

The problem we had, to keep the units at the MSD from the *nearest* opponent, was easily solved by letting that opponent be the one setting the potential in the opponent field, rather than the gravity of the whole opponent group (as in the case of summing all potentials). As for the case of bots using planning, we can not see that this really is a problem for them.

4.5.5 On the methodology

We have used the newer version of the ORTS server for the experiments. On the one hand, it allows us to use the latest version of our bot, which of course is implemented to work with the new server. On the other hand, we could not get one of the last years' participants to work with the new server. Since games like these are not transitive in the sense that if player A wins over player B, and player B wins over player C, then player A will not be guaranteed to win over player C, there is a risk that the bot that was left out of these experiments would have been better than our solution. However, the point is that we have shown that a potential field-based player is able to play significantly better than a number of planning-based counterparts. Although we have no reason to believe that the UofA07 bot would be an exception, we do not have the results to back it up.

The order of the different versions used was determined after running a small series of matches with different combinations of improvements added. We then picked them in the order that best illustrated the effects of the improvements.

However, 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 4.8).

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

Table 4.8: *Results from the ORTS Tankbattle 2008 competition.*

4.6 Conclusions and Future Work

We have presented a five step improvement of a potential field based bot that plays the Strategic Combat game in ORTS. By the full improvement we managed to raise the performance from winning less than 7 per cent to winning more than 99 per cent of the games against four of the top five teams at the ORTS tournament 2007. Our bot did also quite easily win the 2008 tournament.

We believe that potential fields is a successful option to the more conventional planning-based solutions that uses e.g. A* in Real Time Strategy games.

In the future, we will report on the application of the methodology described in Paper I to a number of other ORTS games. We will also set up a new series of experiments where we adjust the ability/efficiency trade-off of the bot in real time to increase the player experience.

CHAPTER

FIVE

PAPER IV

Dealing with Fog of War in a Real Time Strategy Game Environment

Johan Hagelbäck & Stefan J. Johansson
Proceedings of 2008 IEEE Symposium on Computational Intelligence and Games (CIG). 2008.

5.1 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 their opponents, typically in some kind of war setting. An 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. The player usually has an isometric birds view perspective of 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 *cheat* in the sense that they get access to complete visibility (perfect information) of the whole game world, including the positions of the opponent units. Cheating is, according to Nareyek (2004), *"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.

We will show how a bot that uses potential fields can be modified to deal with imperfect information, i.e. the parts of the game world where no own units are present are unknown (usually referred to as Fog of War, or FoW). We will also show that our modified bot with imperfect information, named FoWbot, actually not only perform equally good, compared to a version with perfect information (called PIbot), but also that it at an average consumes *less* computational power than its cheating counterpart.

5.1.1 Research Question and Methodology

The main research question of this paper is: *Is it possible to construct a bot without access to perfect information for* RTS *games that perform as well as bots that have perfect information?* This breaks down to:

1. What is the difference in performance between using a FoWbot compared to a PIbot in terms of a) the number of won matches, and b) the number of units and bases left if the bot wins?

2. To what degree will a field of exploration help the FoW bot to explore the unknown environment?

3. What is the difference in the computational needs for the FoWbot compared to the PIbot?

In order to approach the research questions above, we will implement a FoW version of our original PIbot and compare its performance, exploration and processing needs with the original.

5.1.2 Outline

First we describe the domain followed by a description of our Multi-agent Potential Field (MAPF) player. In the next section we describe the adjustments needed to implement a working FoW bot and then we present the experiments and their results. We finish by discussing the results, draw some conclusions and line out possible directions for future work.

5.2 ORTS

Open Real Time Strategy (ORTS) (Buro, 2007a) is a real-time strategy game engine developed as a tool for researchers within 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 state of the game. Clients can then activate their units in various ways by sending commands to them. These commands can be like *move unit A to* (x, y) or *attack opponent unit X with unit A.* All client commands for each time frame are sent in parallel, and executed in random order by the server.

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 focus here on one type of two-player game, *Tankbattle*, which was one of the 2007 ORTS competitions (Buro, 2007a).

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 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. The purpose of them is to make pathfinding and collision detection more complex.

We have in our experiments chosen to use an environment based on the best participants of the last year's ORTS tournament (Buro, 2007b).

5.2.1 Descriptions of Opponents

We refer to Paper I section 2.5.1 for opponent descriptions.

5.3 Multi-agent Potential Fields

Khatib (1986) 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. Later on 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.

Many studies concerning potential fields are related to spatial navigation and obstacle avoidance, see e.g. Borenstein and Koren (1991); Massari et al. (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 has shown promising results. Howard et al. (2002) developed a mobile sensor network deployment using potential fields, and potential fields have been used in robot soccer (Johansson & Saffiotti, 2002; Röfer et al., 2004). Thurau et al. (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.

In Paper I we propose a methodology for creating an RTS game bot based on Multi-agent Potential Fields (MAPF). This bot was further improved in Paper III and it is the improved version that we have used in this experiment.

5.4 MAPF in ORTS

We have implemented an ORTS client for playing Tankbattle games based on Multi-agent Potential Fields (MAPF) following the proposed methodology of Paper I. 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.

5.4.1 Identifying objects

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

5.4.2 Identifying fields

We identified five tasks in ORTS Tankbattle:

- Avoid colliding with moving objects,
- avoid colliding with cliffs, and
- find the enemy,

- destroy the enemy's forces, and
- defend the bases.

The latter task will not be addressed in this study (instead, see Paper III), but the rest lead us to three types of potential fields: *Field of navigation*, *Strategic field*, *Tactical field*, and *Field of exploration*.

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

The field of exploration helps the units to explore unknown parts of the game map. Since it is only relevant in the case we have incomplete information, it is not part of the PIbot that we are about to describe now. More information about the field of exploration is found in Section 5.5.3.

5.4.3 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. We used the following charges in the PIbot:[1]

The opponent units

$$p(d) = \begin{cases} k_1 d, & \text{if } d \in [0, MSD - a[\\ c_1 - d, & \text{if } d \in [MSD - a, MSD] \\ c_2 - k_2 d, & \text{if } d \in]MSD, MDR] \end{cases} \tag{5.1}$$

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

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 5.1: *The parameters used for the generic $p(d)$-function of Equation 5.1.*

Own bases Own bases generate a repelling field for obstacle avoidance. Below in Equation 5.2 is the function for calculating the potential $p_{ownB}(d)$ at distance d (in tiles) from the center of the base.

$$p_{ownB}(d) = \begin{cases} 5.25 \cdot d - 37.5 & \text{if } d <= 4 \\ 3.5 \cdot d - 25 & \text{if } d \in]4, 7.14] \\ 0 & \text{if } d > 7.14 \end{cases} \tag{5.2}$$

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

$$p_{ownU}(d) = \begin{cases} -20 & \text{if } d <= 0.875 \\ 3.2d - 10.8 & \text{if } d \in]0.875, l], \\ 0 & \text{if } d >= l \end{cases} \tag{5.3}$$

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:

$$p_{sheep}(d) = \begin{cases} -10 & \text{if } d <= 1 \\ -1 & \text{if } d \in]1, 2] \\ 0 & \text{if } d > 2 \end{cases} \tag{5.4}$$

Figure 3.2 in Paper II shows a 2D view of the map during a tankbattle game. It shows our agents (green) moving in to attack enemy bases and units (red). Figure 3.3 shows the potential field view of the same tankbattle game. Dark areas has low potential and light areas high potential. The light ring around enemy bases and units, located at maximum shooting distance of our tanks, is the distance our agents prefer to attack opponent units from. It is the final move goal for our units.

5.4.4 Finding the right granularity

Concerning the granularity, we use full resolution (down to the point level) but only evaluate eight directions in addition to the position where the unit is. However, this is done in each time frame for each of our units.

5.4.5 Agentifying the objects

We put one agent in every own unit able to act in some way (thus, the bases are excluded). We have chosen not to simulate the opponent using agents, although that may be possible, it is outside the scope of this experiment.

5.4.6 Constructing the MAS

All of our unit agents are communicating with a common *interface agent* to get and leave information about the state of the game such as to get the position of (visible) opponents, and to submit the actions taken by our units. The bot also has an *attack coordinating agent* that points out what opponent units to attack, if there are several options.

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. The attack coordinator used are identical to the attack coordinator agent described in Paper I section 2.4.6.

5.5 Modifying for the Fog of War

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).

To deal with fog of war we have made some changes to the bot described in Paper III. These changes deal with issues like remember locations of enemy bases, explore unknown terrain to find enemy bases and units, and to remember the terrain (i.e. the positions of the impassable cliffs at the map) even when there are no units near. Another issue is dealing with performance since these changes are supposed to require more runtime calculations than the PIbot. Below are proposed solutions to these issues.

5.5.1 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 visibility range of the base. It means that an enemy base that 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 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, as well as 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.

5.5.2 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:

$$p_{terrain}(d) = \begin{cases} -10000 & \text{if } d <= 1 \\ -5/(d/8)^2 & \text{if } d \in]1, 50] \\ 0 & \text{if } d > 50 \end{cases} \tag{5.5}$$

5.5.3 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 $p_{unknown}$ generates in a point (x, y) is calculated as follows:

1. Divide the terrain tile map into blocks of 4x4 terrain tiles.
2. For each block, check every terrain tile in the block. If the terrain is unknown in ten or more of the 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 Equation 5.6 below.
5. The total potential in (x, y) is the sum of the potentials each block generates in (x, y).

$$p_{unknown}(md) = \begin{cases} (0.25 - \frac{md}{8000}) & \text{if } md <= 2000 \\ 0 & \text{if } md > 2000 \end{cases} \quad (5.6)$$

5.6 Experiments

We have conducted three sets of experiments:

1. Show the performance of FoWbot playing against bots with perfect information.
2. Show the impact of the field of exploration in terms of the detected percentage of the map.
3. Show computational resources needed for FoWbot compared to the PIbot.

5.6.1 Performance

To show the performance of our bot we have run 100 games against each of the top teams NUS, WarsawB, UBC and Uofa.06 from the 2007 years ORTS tournament as well as 100 matches against our PIbot. In the experiments the first game starts with a randomly generated seed, and the seed is increased by 1 for each game played. The same start seed is used for all four opponents.

The experiment results presented in Table 5.2 shows that our MAPF based FoWbot wins over 98% of the games even though our bot has imperfect information and the opponent bots have perfect information about the game world.

We may also see that when PIbot and FoWbot are facing each other, FoWbot wins (surprisingly enough) about twice as often as PIbot. We will come back to the analysis of these results in the discussion.

	FoWbot			PIbot		
Team	Win %	Units	Base	Win %	Units	Base
NUS	100%	29.74	3.62	100%	28.05	3.62
WarsawB	98%	32.35	3.19	99%	31.82	3.21
UBC	96%	33.82	3.03	98%	33.19	2.84
Uofa.06	100%	34.81	4.27	100%	33.19	4.22
Average	98.5%	32.68	3.53	99.25%	31.56	3.47
FoWbot	—	—	—	66%	9.37	3.23
PIbot	34%	4.07	1.81	—	—	—

Table 5.2: *Performance of FoWbot and PIbot in 100 games against five opponents.*

5.6.2 The Field of Exploration

We ran 20 different games in this experiment, each where the opponent faced both a FoWbot with the field of exploration enabled, and one where this field was disabled (the rest of the parameters, seeds, etc. were kept identical).

Figure 5.1 shows the performance of the exploration field. It shows how much area, for both types of bots, that is explored, given how long a game has proceeded. The standard deviation increases with the time since only a few of the games last longer than three minutes.

In Table 5.3, we see that the use of the field of exploration (as implemented here) does not improve the results dramatically. The differences are not statistically significant.

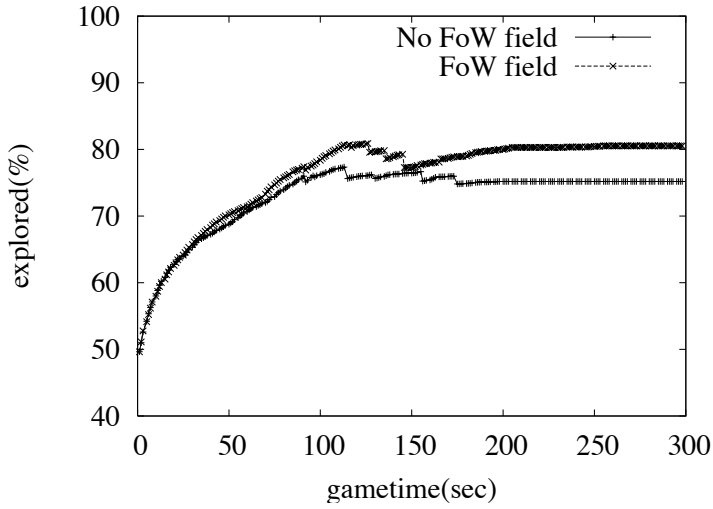

Figure 5.1: *The average explored area given the current game time for a bot using the field of exploration, compared to one that does not.*

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

Table 5.3: *Performance of the FoWbot with and without Field of Exploration (FoE) in 20 matches against NUS.*

5.6.3 Computational Resources

To show the computational resources needed we have run 100 games using the PIbot against team NUS and 100 games with the same opponent using the FoWbot. The same seeds are used in both series of runs. For each game we measured the average time (in milliseconds) that the bot uses in each game frame and the number of own units left. Figure 5.2 shows the average time for both our bots in relation to number of own units left.

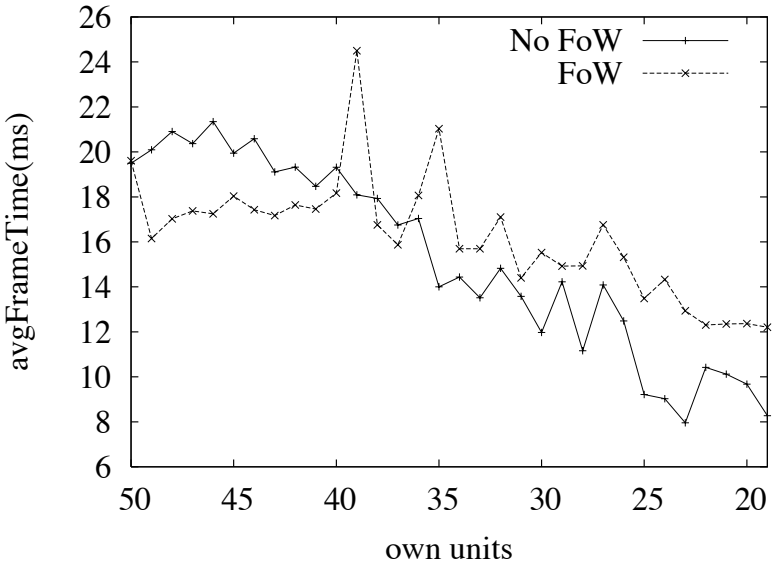

Figure 5.2: *The average frame time used for PIbot and FoWbot against team NUS.*

5.7 Discussion

The performance shows good results, but the question remains: could it be better without FoW? We ran identical experiments which showed that the average winning percentage was slightly higher for the PIbot compared to the FoWbot when they faced the top teams of ORTS 2007, see Table 5.2. We can also see that the number of units, as well as bases left are marginally higher for the FoWbot compared to the PIbot. However these results are not statistically significant.

Where we actually see a clear difference is when PIbot meets FoWbot and surpris-

ingly enough FoWbot wins 66 out of 100 games. We therefore have run a second series of 100 matches with a version of the PIbot where maximum detection range (i.e. the range at which a bot starts to sense the opponents' potential field) was decreased from 1050 to 450. This is not the same as the *visibility range* in the FoWbot (which is just 160). Remember that the FoWbot has a global map agent that helps the units to distribute the positions of visible enemies to units that do not have visual contact with the enemy unit in question. However, the decrease of the maximum detection range in PIbot makes it less prone to perform single unit attacks and the FoWbot only wins 55 out of 100 games in our new series of matches, which leaves a 37% probability that PIbot is the better of the two (compared to 0.2% in the previous case).

In Figure 5.1 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 5.3) 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.

Concerning the processor resources, the average computational effort is initially higher for the PIbot. The reason for that is that it knows the positions of all the opponent units, thus include all of them in the calculations of the strategic potential field. As the number of remaining units decrease the FoWbot has a slower decrease in the need for computational power than the PIbot. This is because there is a comparably high cost to keep track of the terrain and the field of navigation that it generates, compared to having it static as in the case of the PIbot.

This raise the question of whether having access to perfect information is an advantage compared to using a FoWbot. It seems to us, at least in this study, that it is not at all the case. Given that we have at an average around 32 units left when the game ends, the average time frame probably requires more from the PIbot, than from the FoWbot. However, that will have to be studied further before any general conclusions may be drawn in that direction.

Finally some comments on the methodology of this study. There are of course details that could have been adjusted in the experiments in order to e.g. balance the performance of PIbot vs FoWbot. As an example, by setting the detection range in the PIbot identical to the one in the FoWbot and at the same time add the global map agent (that is only used in the FoWbot today) to the PIbot. However, it would significantly increase the computational needs of the PIbot to do so. We are of course eager to improve our bots

as far as possible (for the next ORTS competition 2009; a variant of our PIbot won the 2008 competition in August with a win percentage of 98%), and every detail that may improve it should be investigated.

5.8 Conclusions and Future Work

Our experiments show that a MAPF based bot can be modified to handle imperfect information about the game world, i.e. FoW. Even when facing opponents with perfect information our bot wins over 98% of the games. The FoWbot requires about the same computational resources as the PIbot, although it adds a field of exploration that increases the explored area of the game.

Future work include a more detailed experiment regarding the computational needs as well as an attempt to utilise our experiences from these experiments in the next ORTS tournament, especially the feature that made FoWbot beat PIbot.

CHAPTER

SIX

PAPER V

A Multi-agent Architecture for Real Time Strategy Games

Johan Hagelbäck & Stefan J. Johansson
Submitted for publication.

6.1 Introduction

There are many challenges for a real-time strategy (RTS) bot. The bot has to control a number of units performing tasks such as gathering resources, exploring the game world, hunting down the enemy and defend own bases. In modern RTS games, the number of units can in some cases be up to several hundred. The highly dynamic properties of the game world (e.g. due to the large number of moving objects) make navigation sometimes difficult using conventional pathfinding methods.

There has been several attempts to build Multi-agent architectures for different kinds of games. Kraus and Lehmann (1995) created a multi-agent architecture of heterogeneous agents that played the board game *Diplomacy* (including the negotiations with the opponent players) and MAS in board games in general has been studied by Johansson (2006).

Artificial Potential Fields, an area originating from robotics, has been used with some success in video games. Thurau et al. (2004b) has developed a game bot which learns behaviours in the First-Person Shooter game Quake II through imitation. The behavious are represented as attractive potential fields placed at interesting points in the

game world, for example choke points or areas providing cover. The strength of the fields are increased/decreased by observing a human player.

6.1.1 Multi-agent Potential Fields

In Paper I we proposed a methodology for designing a multi-agent potential fields (MAPF) based bot in a real-time strategy game environment. The methodology involves the following six steps: *i) Identifying the objects, ii) Identifying the fields, iii) Assigning the charges, iv) Deciding on the granularities, v) Agentifying the core objects* and *vi) Construct the* MAS *architecture.* For further details on the methodology, we refer to the original description.

6.1.2 Outline

We start by describing the environment and the scenario used (Section 6.2). In Section 6.3, we apply the methodology in the chosen scenario, and in Sections 6.4–6.6 we describe the experiments, discuss the results and list some directions for future work.

6.2 ORTS

Open Real Time Strategy (ORTS) (Buro, 2007a) is a real-time strategy game engine developed as a tool for researchers within AI in general and game AI in particular. ORTS use a client-server architecture with a game server and players connected as clients. 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.

In previous work (see Paper I and Paper III) we used the proposed methodology to develop a MAPF based bot for the quite simple game type *Tankbattle*. Here, we extend the work to handle the more complex Full RTS game (Buro, 2007a). In this game, two players start with five workers and a control center each. The workers can be used to gather resources from nearby mineral patches, or construct new control centers, barracks or factories. A control center serves as the drop point for resources gathered by workers, and it can produce new workers as well. Barracks are used to construct marines; a light-weight combat unit. If a player has at least one barrack, it can construct a factory. Factories are used to construct tanks; heavy combat units with long firerange. A player wins by destroying all buildings of the opponent. 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.

Both games are part of the annual ORTS tournament organised by the University of Alberta.

6.3 MAPF in a Full RTS Scenario

We have implemented a MAPF based bot for playing the Full RTS game in ORTS following the proposed steps presented briefly in Section 6.1. Since this work extends previous research on MAPF based bots (and the space limitations prevents us from describing everything in detail), we will concentrate this on the additions we have made. For the details about the MAPF methodology and the Tankbattle scenario, we refer to Paper I and Paper III.

6.3.1 Identifying objects

We identify the following objects in our application: *Workers, Marines, Tanks, Control centers, Barracks, Factories, Cliffs, Sheep,* and *Minerals.* Units and buildings are present on both sides.

6.3.2 Identifying fields

In the Tankbattle scenario we identified four tasks: Avoid colliding with moving objects, Hunt down the enemy's forces, Avoid colliding with cliffs, and Defend the bases (see Paper I). In the Full RTS scenario we identified the following additional tasks: Mine resources, Create buildings, Train workers and marines, Construct tanks, and Explore the game world. The tasks are organised into the following types of potential fields: *Field of Navigation*. This field contains all objects that have an impact on the navigation in the game world: *terrain, own units and buildings, minerals* and *sheep.* The fields are repelling to avoid that our agents collide with obstacles.

Strategic Field. This field contains the goals for our agents and is an attractive field, different for each agent type. Tanks has attractive fields generated by opponent units and buildings. Workers mining resources has attractive fields generated by mineral patches (or if they cannot carry anymore, the base where it can drop them).

Field of Exploration. This field is used by workers assigned to explore the game world and attract them to unexplored areas.

Tactical field. The purpose of the tactical field is to coordinate movements between our agents. This is done by placing a temporary small repelling field at the next movement position for an agent. This prevents own units from moving to the same location if there are other routes available.

Field of spatial planning. This field helps us finding suitable places on the map to construct new buildings such as bases, barracks and factories at.

6.3.3 Assigning charges and granularity

Each game objective that has an effect on navigation or tactics for our agents has 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 was hand crafted. 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. Below is a detailed description of each field.

The opponent units. Opponent units, tanks marines and workers, generate different fields depending on agent type and internal state. In case of own attacking units, tanks and marines, opponent units generate attracting symmetric surrounding fields where the highest potentials are at radius equal to the maximum shooting distance, MSD from the enemy unit.

Own buildings. Own buildings, control centers barracks and factories, generate repelling fields for obstacle avoidance. An exception is in the case of workers returning minerals to a control center. In this case control centers generate an attractive field calculated using Equation 6.2. The repelling potential $p_{ownB}(d)$ at distance d from the center of the building is calculated using Equation 6.1.

$$p_{ownB}(d) = 1.429 \cdot \begin{cases} 6 \cdot d - 258 & \text{if } d <= 43 \\ 0 & \text{if } d > 43 \end{cases} \tag{6.1}$$

$$p_{attractive}(d) = \begin{cases} 240 - d \cdot 0.32 & \text{if } d <= 750 \\ 0 & \text{if } d > 750 \end{cases} \tag{6.2}$$

Minerals. Minerals generate two different types of field; one attractive field used by workers mining resources and a repelling field that is used for obstacle avoidance. The potential $p_{attractive}(d)$ at distance d from the center of a mineral is calculated using Equation 6.2. In the case when minerals generate a repelling field, the potential $p_{mineral}(d)$ at distance d from the center of a mineral is calculated as:

$$p_{mineral}(d) = 1.429 \cdot \begin{cases} -20 & \text{if } d <= 8 \\ 20 - 2 \cdot d & \text{if } d \in]8, 10] \end{cases} \tag{6.3}$$

Field of exploration. The field of exploration is a field with attractive charges at the positions in the game world that need to be explored. First we calculate an importance value for each terrain tile in the game world to find a position to explore. This process is described below. Once a position is found, we use the Field of Navigation, Equation 6.4, to navigate to the spot. This approach seems to be more robust than letting all unexplored areas generate attractive potentials. In the latter case explorer units tend to get stuck somewhere in the middle of the map due to the attractive potentials generated from unexplored edges around the game world.

$$p_{navigation}(d) = \begin{cases} 150 - d * 0.1 & \text{if } d <= 1500 \\ 0 & \text{if } d > 1500 \end{cases} \quad (6.4)$$

The importance value for each tile is calculated as follows:

1. Each terrain tile (16x16 points) is assigned an explore value, $E(x, y)$, initially set to 0.
2. In each frame, $E(x, y)$ is increased by 1 for all passable tiles.
3. If a tile is occupied by one or more of our own units in the current frame, its $E(x, y)$ is reset to 0.
4. Calculate an importance value for each tile using Equation 6.5. The distance d is the distance from the explorer unit to the tile.

$$importance(x, y, d) = 2.4 \cdot E(x, y) - 0.1 \cdot d \quad (6.5)$$

Pick the tile of the greatest importance (if there are several equally important, pick one of them randomly), and let that generate the field.

Base building. When a worker is assigned to construct a new building, a suitable build location must first be found. Section 6.3.4 describes the method we use to find the location. Once a location is found, the potential $p_{builder}(d)$ at distance d from the position to build at is calculated using the Field of Navigation (see Equation 6.4).

As in the Tankbattle scenario described in Paper III, we use a granularity of 1x1 points in the game world for potential fields, and all fields are updated every frame.

6.3.4 The agents of the bot

Each own unit (worker, marine or tank) is represented by an agent in the system (see Paper III for a full description of these agents). The multi-agent system also contains a number of agents not directly associated with a physical object in the game world. The purpose of these agents is to coordinate the unit agents to a high-level plan rather than letting them act independently. Below follows a more detailed description of each agent.

CommanderInChief

The CommanderInChief agent is responsible for making an overall plan for the game, called a battleplan. The battleplan contain the order of creating units and buildings, for example start with training 5 workers then build a barrack. It also contain special actions, for example sending units to explore the game world. When one post in the battleplan is

completed, the next one is executed. If a previously completed post no longer is satisfied, for example a worker is killed or a barrack is destroyed, the CommanderInChief agent takes the necessary actions for completing the previous post before continue executing the current one. A new post is executed when all previous posts are satisfied, and when there are enough resources available. The battleplan is based on the ideas of subsumption architectures (see Brooks (1986)) shown in Figure 6.1. Note that all workers, unless ordered to do something else, are gathering resources.

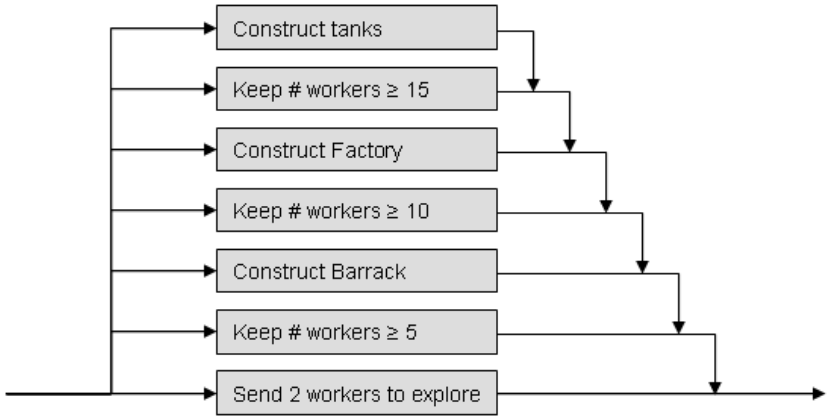

Figure 6.1: *The subsumption hierarchy battleplan used by our bot.*

CommanderInField

The CommanderInField agent is responsible for executing the battleplan generated by the CommanderInChief. It is responsible for setting the goals for each unit agent, and change the goals during the game if necessary (for example use a worker agent currently gathering resources to construct a new building, and to have the worker go back to resource gathering after the building is finished). The CommanderInField agent has two additional agents to help it with the execution of the battleplan; GlobalMapAgent and AttackCoordinator. These are described below.

GlobalMapAgent

In ORTS a data structure with the current game world state is sent each frame from the server to the connected clients. The location of buildings are however only included in the data structure if an own unit is within visibility range of the building. It means that an enemy base that 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 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, as well as distributes the positions of detected enemy units to all the own units.

AttackCoordinator

The purpose of the attack coordinator agent is to optimize attacks at enemy units. The difference between using the coordinator agent compared to attacking the most damaged unit within fire range (which seemed to be the most common approach used in the 2007 years' ORTS tournament) is best illustrated with an example. A more detailed description of the attack coordinator can be found in Paper I.

In Figure 6.2 the own units A, B and C deals 3 damage each. They can all attack opponent unit X (X can take 8 more damage before it is destroyed) and unit A can also attack unit Y (Y can take 4 more damage before it is destroyed). If an *attack the weakest* strategy is used, unit A will attack Y, and B and C will attack X with the result that both X and Y will survive.

By letting the coordinator agent optimise the attacks, all units are coordinated to attack X, which then is destroyed and only Y will survive.

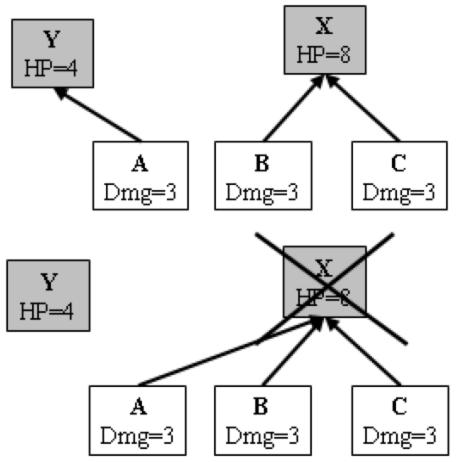

Figure 6.2: *Attacking the most damaged unit within the fire range (upper) vs. Optimise attacks to destroy as many units as possible (lower).*

SpatialPlanner

To find a suitable location to construct new buildings at, we use a special type of field. The field is only used to find a spot to build at; once it has been found by the Spatial Planning Agent, a worker agent use the Field of Navigation (see Equation 6.4) to move to that spot.

Below follow equations used to calculate the potential game objects generate in the find build spot field.

Own buildings. Own buildings generate a field with an inner repelling area (to avoid construct buildings too close to each other) and an outer attractive area (for buildings to be grouped together). Even though the size differs somewhat between buildings for simplicity we use the same formula regardless of the type of building. The $p_{ownbuildings}(d)$ at distance d from the center of an own building is calculated as:

$$p_{ownbuildings}(d) = \begin{cases} -1000 & \text{if } d <= 115 \\ 230 - d & \text{if } d \in]115, 230] \\ 0 & \text{if } d > 230 \end{cases} \tag{6.6}$$

Enemy buildings. Enemy buildings generate a repelling field. The reason is of course that we do not want own buildings to be located too close to the enemy. The $p_{enemybuildings}(d)$ at distance d from the center of an own building is calculated as:

$$p_{enemybuildings}(d) = \begin{cases} -1000 & \text{if } d <= 150 \\ 0 & \text{if } d > 150 \end{cases} \tag{6.7}$$

Minerals. It is not possible to construct buildings on top of minerals; therefore they too have to be repelling. The $p_{mineral}(d)$ at distance d from the center of a mineral is calculated using Equation 6.8. The field is slightly attractive outside the repelling area since it is beneficial to have bases located close to resources.

$$p_{mineral}(d) = \begin{cases} -1000 & \text{if } d <= 90 \\ 5 - d \cdot 0.02 & \text{if } d \in]90, 250] \\ 0 & \text{if } d > 250 \end{cases} \tag{6.8}$$

Impassable terrain. Cliffs generate a repelling field to avoid workers trying to construct a building too close to a cliff. The $p_{cliff}(d)$ at distance d from the closest cliff is calculated as:

$$p_{cliff}(d) = \begin{cases} -1000 & \text{if } d <= 125 \\ 0 & \text{if } d > 125 \end{cases} \tag{6.9}$$

Game world edges. The edges of the game world have to be repelling as well to avoid construction of buildings outside the map. The $p_{edge}(d)$ at distance d from the closest edge is calculated as:

$$p_{edge}(d) = \begin{cases} -1000 & \text{if } d < 90 \\ 0 & \text{if } d >= 90 \end{cases} \tag{6.10}$$

To find a suitable location to construct a building at, we start with calculating the total buildspot potential in the current position of the assigned worker unit. In the next step we calculate the buildspot potential in points at distance 4 from the location of the worker, in next step distance 8, and continue up to distance 200. The position with the highest buildspot potential is the location to construct the building at. If the location is, for example due to massive terrain, still not suitable to build at a new calculation is performed once the worker reaches the destination.

6.4 Experiments

We used the ORTS tournament of 2008 as a benchmark to test the strength of our bot. The number of participants in the Full RTS game was unfortunately very low, but the results are interesting anyway since the opponent team from University of Alberta has been very competitive in earlier tournaments. The UOFA bot uses a hierarchy of commanders where each major task such as gathering resources or building a base is controlled by a dedicated commander. The Attack commander, responsible for hunting down and destroy enemy forces, gather units in squads and uses A* for the pathfinding. The results from the tournament are shown in Table 6.1. Our bot won 82.5% of the games against the opponent team over 2x200 games (200 different maps where the players switched sides).

Team	Win %	Wins/games	DC
Blekinge	82.5%	(330/400)	0
Uofa	17.5%	(70/400)	3

Table 6.1: *Results from the ORTS tournament of 2008. DC is the number of disconnections due to client software failures.*

6.5 Discussion

There are several interesting aspects here.

- First, we show that the approach we have taken, to use Multi-agent potential fields, is a viable way to construct highly competitive bots for RTS scenarios of medium complexity. Even though the number of competitors this year was very low, the opponent was the winner (89 % wins) of the 2007 tournament. Unfortunately, server updates has prevented us from testing our bot against the other participant of that year, but there are reasons to believe that we would outperform that solution too (although it is not sure, since the winning relation between strategies in games is not transitive, see e.g. Rock, Paper Scissors (deJong, 2004)). We argue though that the use of open tournaments as a benchmark is still better than if we constructed the opponent bots ourselves.

- Second, we combine the ideas of using a role-oriented MAS architecture and MAPF bots. By doing so, we may argue that the ideas about role-oriented MAS that originally was applied to a board game (Kraus & Lehmann, 1995), also works in a real time setting.

- Third, we introduce (using the potential field paradigm) a way to place new buildings in RTS games.

6.6 Conclusions and Future Work

We have constructed an ORTS bot based on both the principles of role-oriented MAS and Multi-agent Potential Fields, able to play a full RTS game. It outperforms the competitor by winning more than 80% of the games in an open tournament where it participated.

Future work will include to generate a battleplan for each game depending on the skill and the type of the opponent it is facing. If, for example, an own factory is destroyed it might not be the best option to directly construct a new as our bot does now. A better strategy might be to construct marines and/or move attacking units back to the base to get rid of the enemy units before constructing a new factory. We believe that a number of details in the higher level commander agents may improve in the future versions when we better adapt to the opponents.

REFERENCES

Alexander, B. (2006). Flow Fields for Movement and Obstacle Avoidance. In *AI Game Programming Wisdom 3.* Charles River Media.

Arkin, R. C. (1987). Motor schema based navigation for a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation.*

Borenstein, J., & Koren, Y. (1989). Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*.

Borenstein, J., & Koren, Y. (1991). The vector field histogram: fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*.

Brooks, R. (1986). A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation* (p. RA-2(1):1423).

Buro, M. (2007a). *ORTS — A Free Software RTS Game Engine.* Retrieved 2009-02-23, from `http://www.cs.ualberta.ca/~mburo/orts/`

Buro, M. (2007b). *ORTS RTS game AI competition, 2007.* Retrieved 2009-02-23, from `http://www.cs.ualberta.ca/~mburo/orts/AIIDE07`

Buro, M. (2008). *ORTS RTS game AI competition, 2008.* Retrieved 2009-02-23, from `http://www.cs.ualberta.ca/~mburo/orts/AIIDE08`

Buro, M., & Furtak, T. (2004). RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS).*

Co, K. (2007). *The Evolution of Real Time Strategy Video Games: The Top 10 Most Revolutionary RTS Games.* Retrieved 2009-02-23, from `http://www.slideshare.net/wuzziwug/rts-evolution`

deJong, E. (2004). Intransitivity in coevolution. In Parallel Problem Solving from Nature - PPSN VIII. In *volume 3242 of Lecture Notes in Computer Science.* Springer.

Demyen, D., & Buro, M. (2008). Fast Pathfinding Based on Triangulation Abstractions. In *AI Game Programming Wisdom 4.* Charles River Media.

Higgins, D. (2002). How to Achieve Lightning-Fast A*. In *AI Game Programming Wisdom.* Charles River Media.

Howard, A., Matarić, M., & Sukhatme, G. (2002). Mobile sensor network deploy-

ment 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).*

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

Johansson, S. J., & Saffiotti, A. (2002). An electric field approach to autonomous robot control. In *RoboCup 2001.* Springer Verlag.

Johnson, G. (2006). Avoiding Dynamic Obstacles and Hazards. In *AI Game Programming Wisdom 2.* Charles River Media.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research.*

Khatib, O. (2004). Human-like motion from physiologically-based potential energies. In J. Lenarcic & C. Galletti (Eds.), *On Advances in Robot Kinematics* (pp. 149–163). Kluwer Academic Publishers.

Koenig, S., & Likhachev, M. (2006). Real-Time Adaptive A*. In *Proceedings of Autonomous Agents and Multi-agent Systems (AAMAS).*

Kraus, S., & Lehmann, D. (1995). Designing and building a negotiating automated agent. *Computational Intelligence*, 11(1):132171.

Mamei, M., & Zambonelli, F. (2004). 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.

Massari, M., Giardini, G., & Bernelli-Zazzera, F. (2004). 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.*

Nareyek, A. (2004). AI in computer games. In *ACM Queue 1(10)* (p. 58-65).

Niederberger, C., & Gross, M. H. (2003). *Towards a Game Agent* (Tech. Rep.). CS Technical Report #377.

Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., & Phillips, J. (2008). GPU computing. In *Proceedings of IEEE* (Vol. 96, p. 879-899).

Reynolds, J. (2002). Tactical Team AI Using a Command Hierarchy. In *AI Game Programming Wisdom.* Charles River Media.

Röfer, T., Brunn, R., Dahm, I., Hebbel, M., Homann, H., J
ngel, M., et al. (2004). *GermanTeam 2004 - the German National RoboCup team.*

Sun, X., Koenig, S., & Yeoh, W. (2008). Generalized Adaptive A*. In *Proceedings of Autonomous Agents and Multi-agent Systems (AAMAS).*

Thurau, C., Bauckhage, C., & Sagerer, G. (2004a). Imitation learning at all levels of game-AI. In *Proceedings of the International Conference on Computer Games, Artificial Intelligence, Design and Education.* University of Wolverhampton.

Thurau, C., Bauckhage, C., & Sagerer, G. (2004b). Learning human-like movement behavior for computer games. In *Proceedings of the 8th International Conference*

on the Simulation of Adaptive Behavior (SAB'04).

Tomlinson, S. L. (2004). The long and short of steering in computer games. *International Journal of Simulation Systems, Science and Technology*.

Vadakkepat, P., Tan, K. C., & Ming-Liang, W. (2000). Evolutionary artificial potential fields and their application in real time robot path planning. In *Proceedings of the 2000 Congress on Evolutionary Computation* (pp. 256–263). IEEE Press.

vanLent, M., Laird, J., Buckman, J., Hartford, J., Houchard, S., Steinkraus, K., et al. (1999). Intelligent agents in computer games. In *Proceedings of AAAI.*

Wirth, N., & Gallagher, M. (2008). An Influence Map Model for Playing Ms. Pac-Man. In *Proceedings of 2008 IEEE Symposium on Computational Intelligence and Games (CIG).*