NTNU – Trondheim
Norwegian University of
Science and Technology

Applications of artificial potential fields for real time strategy games

Troop formations and movements used trained potential functions

Finn Robin Kåveland Hansen

Master of Science in Computer Science
Submission date: June 2012
Supervisor: Helge Langseth, IDI
Co-supervisor: Anders Kofod-Petersen, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Applications of artificial potential fields for real time strategy games

Troop formations and movements using trained potential functions

Master thesis by
Robin Kåveland Hansen

Supervised by
Helge Langseth
Anders Kofod-Petersen

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer Science
Norwegian University of Technology and Science

June 26, 2012

Abstract

This thesis describes the effort of adapting potential field methods towards managing groups of units in real-time strategy game like environments. The focus is on discovering the suitability of this technology to create realistic group behaviour and whether it can be adapted to use learning for this purpose.

A flexible simulation system for units that are controlled by potential field methods in environments similar to those found in real-time strategy games is developed. This is then used to conduct experiments where different types of group behaviours are sought after. Stochastic optimization techniques are applied in an attempt to create and optimize desired group behaviour. The results are discussed and possible future directions for this research are proposed.

Sammendrag

Denne oppgaven beskriver arbeidet ved å bruke potensialfelt-baserte metoder for å styre grupper av enheter i miljøer som likner på real-time strategispill. Fokuset settes på å oppdage hvor passende teknologien er for å oppnå realistisk oppførsel av grupper og om den kan bli tilpasset til å bruke læringsteknikker.

Et fleksibelt simulasjonssystem for enheter som blir kontrollert med potensialfelt-baserte metoder i slike miljøer blir utviklet. Dette blir brukt til å utføre eksperimenter som har som mål å få grupper av enheter til å adlyde bevegelsesmønstre som er nyttige i strategispill. Stokastiske optimiseringsteknikker blir benyttet for å skape og forbedre potensialfunksjoner som gir ønskede bevegelsesmønstre. Oppgaven avslutter med å diskutere resultatene som blir funnet og se på mulige interessante retninger for videre undersøkelser av funnene.

Acknowledgements

Thanks to Helge Langseth for pushing me in the right direction when I was headed nowhere and for limitless patience when I needed advice.

Thanks to Anders Kofod-Petersen for valuable feedback on this thesis and advice on how to perform research.

Thanks to my family for giving me your best my entire life. You mean more to me than I can express with any amount of words.

Thanks to Ronja Eline Kristensen for the patience, support and taking care of me when I get too distracted by my thesis to do it myself.

Thanks to all my friends and colleagues. I have been nothing short of incredibly lucky in getting to know so many wonderful people.

Robin Kåveland Hansen
Trondheim, June 26, 2012

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

The computer game industry is a billion dollar industry in the US alone [Williams, 2002], and computer games are quickly becoming one of the most popular forms of home entertainment in the world. Real-time strategy games is a subgenre of computer games that present several difficult problems to solve. They are noted for being complex approximations to the real world. Most computer players of these games either cheat by using their game engine access to obtain additional resources, units or information - or they are no match for expert level human players. Because of their complexity as simplified simulations of the real world, creating good computer players for these games can potentially be applied to many real world problems [Laird and VanLent, 2001]. Because they are common, they present an attractive arena for testing artificial intelligence techniques extensively with relatively little need for expensive equipment [Buro and Furtak, 2003]. The concepts of real-time strategy games are discussed in some depth in Section 2.1.

One of the areas in which computer players can easily beat human players is the amount of actions they can perform per time unit. This is called *APM* (short for actions per minute) in the gaming community. Because human players need mechanical skills such as using a keyboard or a mouse to issue instructions, they can not hope to match the speed with which a computer can issue instructions to a game. This has not been leveraged particularly well in current games. Almost no commercial games utilize learning systems or indeed any cutting edge artificial intelligence techniques from academia. There is a large gap to be filled in discovering which techniques that can be applied with success in real-time strategy games. Currently, most games employ one of or a combination of rule-based systems, scripted players and finite state machines to govern computer player behaviour.

One way to utilize the capability of issuing many instructions in small time is to control single units in a smart manner. Where a human player will have to issue orders to groups of units, a computer player can issue orders to each unit individually, allowing for much more fine grained control of their movements. This need not only be leveraged to create better computer players. It can also allow humans to issue orders to a group of units that they then execute in a novel and intelligent way, creating a more immersive and realistic gaming experince. Seen this way, a group of units in a real-time strategy game is a multi-agent system that receives instructions from the outside and decides how to execute the orders internally.

Potential field methods originated to control robots in static configuration spaces in the late 1980s [Khatib, 1986]. They use the powerful metaphor that obstacles should exert a repulsive force on the robot and goals should exert attractive forces on it. [Reynolds, 1987] shows how simple rules for an individual in a large group can cause the group as a whole to exhibit interesting and realistic behaviours. [Reif and Wang, 1999] adapts the idea to potential field methods to create distributed behaviour in multi-agent systems governed by potential functions. Potential functions are created by following a methodical approach of how to define them such that they result in the desired behaviour. [Vadakkepat et al., 2000] shows that it is possible to apply evolutionary computation techniques to potential functions and that this results in desired behaviour. Potential field methods are attractive because of their elegance, simplicity and efficiency. They are discussed in depth in Section 2.2.

1.2 Goals and research questions

This thesis is focused on filling the niche of micro-management in the context of real-time strategy games. This is an area in which a computer player should be able to function better than human players, but at present this is not the case. Further, it is also the case that most games use their own implementation of rules, scripting or finite state machines to govern individual units or groups of units. This type of specification is tedious, error-prone and can not easily be made to handle unanticipated environments and situations. This thesis is focused towards discovering whether it is possible to do this better. The two high level research questions are:

1. Can potential field methods be applied as a good control mechanism for groups of units in a real-time strategy game?

2. Can learning be applied to potential field methods in order to provide a good control mechanism for groups of units in a real-time strategy game?

A good control mechanism for groups of units needs to satisfy several properties. This thesis will focus primarily on being able to create coherent formations of units that correspond to classical military formations that are applied to win tactical battles in real-time strategy games. The control mechanism clearly needs to be able to accomodate unit groups of different sizes. Furthermore it needs to be able to work in different environments and with units that have non-uniform properties. For this work, the main property that will vary amongst units is their movement speed.

Learning is very desirable because it can reduce the complexity of setting up the control mechanism to work properly and simplify the process of adapting it to new situations. Furthermore, a capability for learning is one of the most desirable goals for computer opponents in real-time strategy games because it provides an excellent way for the game to adapt difficulty level to the skill level of the human opponent. This is highly desirable in order to increase replay value for the game.

The stated goals extracted from these research questions are:

1. Create classical military formations that can adapt to different amounts of units and different environments in a real-time strategy game like system, using potential field setups crafted manually as the control mechanism.

2. Create classical military formations that can adapt to different amounts of units and different environments in a real-time strategy game like system, using optimization techniques to create potential functions for the control mechanism.

1.3 Research method

The research method that will be used for this thesis is to develop a simulation system that can approximate real-time strategy game environments and control simulated units through the use of potential field methods.

In order to test the suitability of potential field based methods as a control mechanism for groups of units, experiments will be performed that attempt to create classical military formations. Each formation will be tested for robustness by introducing additional complexity into its environment.

In order to test whether trained potential fields can be used as a control mechanism for groups of units, experiments will be performed where well-known optimization techniques are applied in order to create military formations. Each formation will be tested for robustness by introducing additional complexity into its environment.

Performance measurements will be used to provide a numerical context with which to evaluate results. Visual inspection will be used to discuss the characteristics of the formations that are created.

1.4 Thesis Structure

Chapter 2 sets the context for this thesis. It starts by discussing the domain of real-time strategy games and prior research that is relevant to this thesis in Section 2.1. In Section 2.2 artificial potential field methods are introduced. Different methods are discussed in some depth and we take a look at the limitations and problems with the method. We finish this section by discussing prior research in group behaviour using potential fields. Section 2.3 ends the chapter by looking at some techniques that can be used to train potential functions.

Chapter 3 documents the experimental setup used for the work in this thesis. That includes Section 3.1, which names the requirements for a potential field simulation system and Section 3.2 which discusses the design and implementation used for the simulation system that was created for this thesis. Section 3.3 shows the basic experimental setup that was used for all experiments performed for this thesis, including a discussion of the simulation environment and how optimization techniques were applied to create potential functions.

Chapter 4 documents the experiments that were performed for this work. Section 4.1 documents the creation of a moving line formation, a box formation and a wedge formation through the use of artificial potential fields for distributed behaviour. Section 4.2 tests the developed formations in environments and circumstances that are more complex than the environments and circumstances they were created in.

Chapter 5 concludes this thesis. Section 5.1 presents a comparison of the environments used in the experiments and typical properties of environments in real-time strategy games. Section 5.2 is a review of the experimental results that sets the stage for Section 5.3, which discusses whether the research goals for this thesis have been met. Section 5.4 ends this thesis by looking at the possible directions in which to take this work in the future.

The appendix is a guide to the files that accompany this thesis, including source code, videos and experimental data.

Chapter 2

Theory and Background

2.1 Real-time strategy games

Concepts of real-time strategy games

Real-time strategy is a genre of computer games where the participants manage units and structures under their control to dominate areas of the map in which the game takes place, or destroy their opponents' units and structures. Very often resource and information gathering are paramount, and many games feature several different types of resources that need to be accumulated to expand armies and bases. Typically the map in which the game is played is covered in *fog of war*, a shroud that covers unexplored areas of the map and makes enemy operations in the area invisible.

Games take place in game worlds usually called maps. Maps are either auto-generated or designed by engineers to have different base locations for players, resource nodes, obstacles and often different types of terrain or elevation. Real-time strategy games progress in real-time, which means that quick decision making is important, because there is rarely much time to deliberate and make perfect calls.

Real-time strategy games usually have different types of *units* with different properties and capabilities. Some very typical properties will be their maximum speed, whether the are ground-based or whether they fly, their armour and their attack range and damage. Units are produced by certain *structures* or become available when certain structures are built. Very often structures enable upgrades to units to be purchased or researched. Together these concepts form the *tech tree* - each new technology that becomes available may unlock several others. For example, building a tank factory will enable building tanks and possibly upgrading

Figure 2.1: StarCraft, blockbuster real-time strategy game

their armour.

When people talk about real-time strategy games, they will often refer to four different concepts that each designate an area in which a player needs to perform well. The most natural of these is *strategy* - this is the plan a player has on how to win the game. A strategy need not be static, it can be a plan for the first 2 or 3 minutes of a game and then it will need to be adjusted as more knowledge about the environment and the opponents become available. The strategy is typically formed by a player's experience and their knowledge of the tech tree, the map and their opponent. The execution of a strategy is often spoken about as *macro management*. This could entail gathering enough resources to execute the strategy, to complete structures at the correct timings, to keep producing units that gather resources and to purchase upgrades. This means that a strategy is a mental model of how the game should proceed and macro management refers to the mechanical skill necessary to achieve it.

Similarly there is a divide between the *tactic* and the control of units in a manner such that they conform to it, aptly called *micro management*. A tactic is the mental model of how to approach a battle, given the units that have been provided by the execution of the strategy. This could for example be to surround an enemy

group of units with ones own units. Micro management is the mechanical aspect of this, that is to provide the units with orders that enables them to complete the tactic. A tactic need not be static either, if a player is losing a battle, they may want to salvage what they can from it and try to escape with some units, or they may want to just do as much damage as possible.

These four different concepts interact with each other. A player that excels at macro management but is not very good at micro management would perhaps employ a strategy that has less stringent requirements for micro management. A player who has a great strategy may still lose against a less clever player with better mechanical skills. Skill at the mechanical aspects of the game enables a player to use more complex and demanding strategies and tactics.

Suitability for AI research

Real-time strategy games constitute an excellent test-bed for AI research. They present large, complex environments that attempt to approximate the real world. Decisions must be made quickly, and the players must plan ahead far into the future. They must reason at many different levels of abstraction and they must learn, if they are to perform well. Often, important decisions must be made quickly, with imperfect information. This makes it impossible to approach a real-time strategy game like one would approach many of the games where computers can outperform world-class human players, like chess and checkers. Enumerating all possibilities is simply put impossible, or in simple games, prohibitive because of the computational complexity.

Many classical AI problems arise in real-time strategy games. The following is by no means an exhaustive list of desirable problems to solve for an agent that plays a real-time strategy game (adapted from [Buro and Furtak, 2004]):

Real-time planning
: A strategy needs to be developed and adjusted throughout the game, with little time to calculate it.

Opponent modelling
: Humans adjust to changes in their opponent's strategies and computer players that do not do this are at a severe disadvantage.

Terrain analysis
: Humans are able to utilise choke points and other terrain traits to maximise the efficiency of their troops. Additionally they are capable at judging which parts of the map that are important.

Resource management
: All real-time strategy games have trade offs where a bigger military force can

be obtained early at the expense of research or economy. An agent needs to manage his resources well to reach a good result against a human player.

Path-finding
Path-finding is a simple problem that is made more complicated by a dynamic and hostile environment. The goal may no longer be to find the shortest path, but to find the safest path that does not send forces too far from the agent's home base.

Information retrieval
The agent needs to obtain information, which in itself is a relatively simple task. However due to having limited time and resources it needs to be smart about which information to obtain.

Learning
To ever compete with humans, an agent would be required to learn from the games it plays.

Real-time strategy games are partially observable, stochastic, dynamic, multi-agent environments. This means that they have a lot in common with real world environments - certainly that they are very difficult environments for an agent to master. An artificially intelligent agent that competes with world class human players in real-time strategy games would be very impressive indeed.

Furthermore, real-time strategy games themselves can have many real world usages - including training programs and military simulations. [Laird and VanLent, 2001] and [Buro and Furtak, 2003] make compelling arguments that real-time strategy games provide an excellent venue for creating advanced AI programs.

Game industry artificial intelligence

The game industry has different goals for artificial intelligence than academia. This is because it is an entertainment industry and the primary goal for an AI opponent in a game is to provide an entertaining challenge that is neither too easy nor too hard to beat. Furthermore, the constraints for the resources allowed to be used by an AI are significant - games are complex, and components such as networking, physics simulations and graphics consume most of the computational resources a game can use. As such, most games in the industry use simple and efficient techniques to implement the computer players. Some common techniques include:

AI scripting
AI scripting is very common and it typically means that the player follows a set sequence of steps that describe how it should behave. Although a

computer player that uses a script will not be adaptable, it can still have a multitude of scripts it can use. Scripts are typically created by expert level humans and can provide a good challenge for novice players.

Rule Based Systems
Rule based systems are also extremely common. While they are often used in conjunction with scripts, they can also be used on their own. Rule based systems enhance the gaming experience with an opponent that can react in diverse ways - for example it might react to a player attacking it by performing a counter attack on the player's base.

Finite State Machines
Finite State Machines allow for more complex behaviour than RBS. They allow the agent to have different scripts for different situations or phases of the game and different rules for how to react, given their state.

None of these techniques allow for much reasoning - and more importantly, while the computer player using these techniques may be able to react to different types of situations, it is not able to learn. Furthermore, it is reliant on the help of human experts to provide any sort of challenge to human players and even then they may not be particularly good. It is very hard for programmers to anticipate human ingenuity and prepare rules for how to handle all the different strategies that players might come up with against the computer player.

In practice, this works out well enough. Although many games ship with computer players that do not provide challenge to humans for long, most modern computer games allow humans to play against one another. The industry is mainly concerned with providing a believable opponent for human players. Some games create challenging opponents for players by enabling the computer to cheat - letting them to see the entire map, and giving them a resource advantage are the most common ways of doing this. This can be both confusing and annoying for human players and the industry is increasingly moving away from this way of challenging them. Additionally most modern games come with multiplayer modes that let humans play against one another once the computer opponent becomes too boring or easy to play against.

Publications of real-time strategy game artificial intelligence

In recent years, creating agents that play real-time strategy games has become much easier. In the past, it has been very difficult to make agents that play these games - game industry is a billion dollar industry and companies are reluctant to release their source code or APIs that allow people to implement agents that play their games. Because real-time strategy games are incredibly complex, it has not been feasible for researchers to implement and test their own games to use for implementing agents.

Projects like ORTS [Buro, 2003] and BWAPI [BWAPI, 2012] are rapidly changing this. ORTS is an open source game engine that was created to study AI in real-time strategy games. BWAPI is an open source library that can connect to an instance of running game of StarCraft: Brood War, Blizzard Entertainment's massively successful game. Both of these frameworks facilitate creating AI software that play the games. StarCraft: Brood War is one of the most successful games of all time and the human competition here is fierce, even 13 years after its 1998 release.

AIIDE (Artificial Intelligence and Interactive Entertainment) arranges competitions between agents that play StarCraft: Brood War annually and some of the recent entries have achieved impressive results. The Berkely Overmind is one example of a bot that has done well enough to garner attention in the popular press [Huang, 2011]. The Overmind uses several technologies that are not typically used for this purpose in the game industry to great effect. Notably it uses machine learning and artificial potential fields. This combination enabled it to win in the 2010 AIIDE competition, [Weber, 2011].

Surprisingly, it is also possible to do well in the AIIDE competition with very simple concepts. MimicBot is perhaps the most startling example of this - it successfully defeated several much more advanced bots using the simple idea to mimic the opponents' strategies. By focusing on micro-management and tactics, it defeated bots that were much better at reasoning about strategy. This is an important point - reasoning incredibly well about strategy is not much help if you are unable to fight your battles smartly. MimicBot does very clever terrain analysis and its choke-point detection algorithm has been published in [Perkins, 2010].

[Weber et al., 2010] presents a more complicated agent that plays StarCraft. This agent is designed to be able to pursue goals of different granularity at the same time, enabling it to reason both about strategical and tactical level decisions. The authors name this requirement *multi-scale* AI. This works proceeds and culminates in [Weber, 2012].

Some headway has been made in artificial intelligence in other genres of games that could be applied to real-time strategy games in the future. [Orkin, 2004] discusses an approach named *Goal Oriented Action Planning* that has been successfully used in the commercial first person shooter game *F.E.A.R.*. GOAP uses a planning approach that works a lot like *STRIPS* and employs A*-search to guide the planner. [Long, 2007] finds this to enhance the behaviour of non-player characters significantly for first-person shooter games. [Weber et al., 2011] notes some difficulty in adapting this to multi-scale AI situations because of the difficulty in letting the system reason on multiple levels and representing worlds symbolically in a manner that enables GOAP to function well.

2.2 Artificial potential fields

Introduction to Artificial Potential Fields

Artificial potential fields is a technique that was initially developed to solve the robot navigation problem. They were proposed for this purposed by [Khatib, 1986] and have been adapted in many different ways.

Potential fields uses observations from physics where complex, stable motions that have many properties that would be useful in robotics arise from relatively simple laws. By simulating repulsive forces from obstacles and attractive forces from objectives, one could get a heading for a robot that is looking to solve navigation problems.

Potential field functions provide for a highly efficient technique of obstacle avoidance that is also simple to implement. We shall also see that the technique has applications that range far beyond path planning.

Traditional artificial potential fields

Artificial potential fields typically work by having a potential function that takes as input the robot's position and calculates the sum of forces on it. In many cases, it turns out that the results improve when the potential function also takes into account the current velocity of the robot. For our purposes, a potential function is a function that takes the state of the robot R, and its environment E as input and produces the impulse $\vec{a}$ it should set on its actuators at time t:

$$\vec{a}(t) = f(R(t), E(t)) \tag{2.1}$$

For static configuration spaces, $E(t)$ is simply the same state, regardless of its time parameter. Traditional potential functions have this property. The potential field is the gradient of the potential function in the robots configuration space and because the environment is static, it can be entirely precomputed. The robot will then simply have a table of impulses it should set at different positions in the environment. The potential function becomes a function of the robots position $p(t)$ and its static environment E.

$$\vec{a}(t) = f(p(t), E) \tag{2.2}$$

The technology artificial potential fields get the name from the fact that they are negative gradients of potential functions. The gradient of the potential function can show us where the robot would attempt to go in any position in its environment. Namely, it would always go from high potential to low potential. Therefore, the usual representation is a potential field function U and its negative gradient. This is a gradient descent method. Other potential field methods instead will sum the potential functions on the robot and use the result to calculate where it should go. These methods are equivalent in practice.

(2.3) shows a common attractive potential function adapted from [Ge and Cui, 2000]. ξ is a positive scaling factor that determines the strength of the attraction. It is usually the case that $m = 1$ or $m = 2$. This results in a constant or parabolic shape for the magnitude of the pull as a function of the distance $r(p, g) = |g - p|$ from the robot to its goal, respectively. (2.4) is the force that results from setting $m = 2$. An object that emits an attractive potential is often called a goal, target or an attractor.

$$U_{\text{att}}(p, g) = \frac{1}{2}\xi r^m(p, g) \tag{2.3}$$

$$\mathbf{F}_{\text{att}}(p, g) = -\nabla U_{\text{att}}(p, g) = \xi(g - p) \tag{2.4}$$

Constructing a repulsive potential field is very similar in nature. However in this case, rather than multiplying by distance as done in (2.3), one divides by distance such that the repulsion is strong when very near the source of it. Using η as a positive scaling factor for the magnitude of the force, $r(p, o)$ as the minimal distance between the robot and an obstacle and ρ_0 as the distance of influence of the obstacle, one can construct the common potential function U_{rep} as in (2.6). When the distance between the obstacle and the robot is shorter than the influence distance, this results in the force (2.7). An object that emits a repulsive potential is often called an obstacle or a repulser.

$$f(p, o) = \frac{1}{r(p, o)} - \frac{1}{\rho_0} \tag{2.5}$$

$$U_{\text{rep}}(p, o) = \frac{1}{2}\eta f^2(p, o), \text{ If } r(p, o) < \rho_0 \text{ else } 0 \tag{2.6}$$

$$\mathbf{F}_{\text{rep}} = -\nabla U_{\text{rep}}(p, o) = \eta f(p, o)\frac{1}{r^2(p, o)}\nabla r(p, o) \tag{2.7}$$

The total potential field for the robot is obtained by summing up all the fields in its environment. Usually, there is one for the goal F_{att} and one for each obstacle

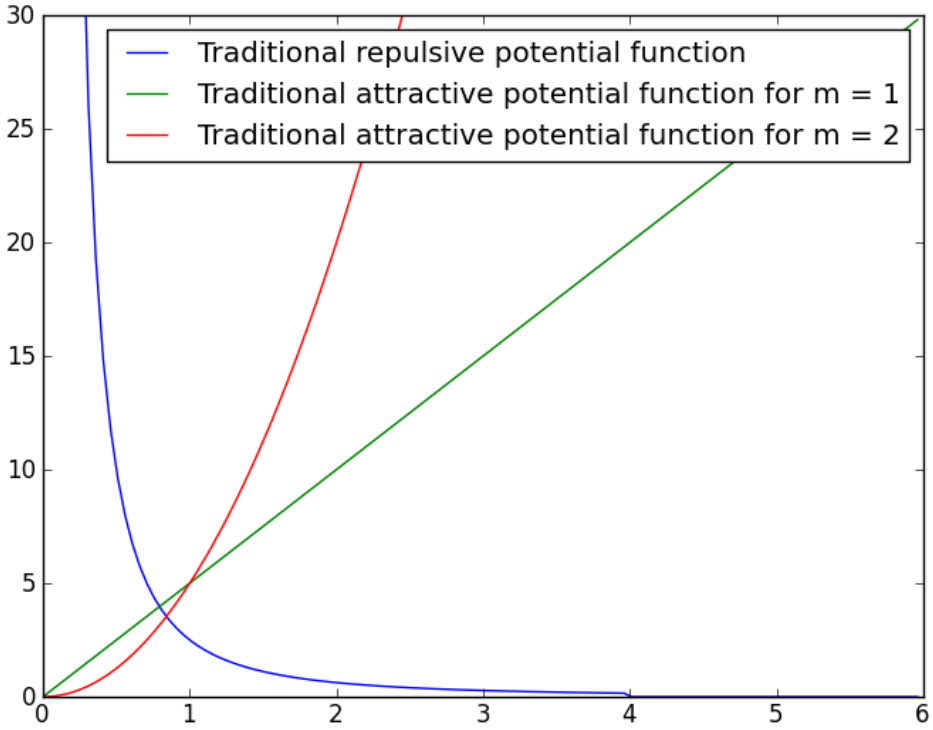

Figure 2.2: Magnitudes of (2.3) for $\xi = 10$ and (2.6) for $\eta = 5, \rho_0 = 4$ for $r \in (0, 6]$

o such that the total is:

$$\mathbf{F}(p) = \mathbf{F}_{\text{att}}(p, g) + \sum_{o} \mathbf{F}_{\text{rep}}(p, o) \tag{2.8}$$

The calculated force vector can then be translated in some way to set the impulses on the robot actuators and produce its acceleration:

$$\vec{a}(t) = t(\mathbf{F}(p(t))) \tag{2.9}$$

For a static environment, this can be precomputed and reused for the duration that the robot is in this environment. The simplest method way to adapt this type of potential in a dynamic environment is to allow the robot to recalculate the potential as often as often as needed. Doing this, only a small area in the environment is usually calculated at a time, maintaining the efficiency of the method.

For dynamic environments, a useful variant of (2.1) is one that also takes into

account both the velocity $\vec{v_r}$ of the robot. For further precision, the velocities of obstacles are used and a potential function that depends on the relative velocity $\vec{v_{ro}}$ between the robot and obstacle o is developed in [Ge and Cui, 2002]. It is shown that using this additional information, more advanced behavior can be produced. The following scenarios are analyzed in depth, using a potential field of the form (2.10): moving target, moving target in environment with obstacles, moving target with moving obstacles.

$$\mathbf{F} = -\nabla U(p_r, v_{ro}, p_o) \tag{2.10}$$

It is shown that adapting potential fields in this way produces good paths for robots in very difficult environments.

Artificial potential fields in robotics

Although there are many advantages to this approach of navigation, problems were also discovered, some of them without obvious solutions. Particle systems can eventually create static configurations, which takes place when each particle gets stuck in a minimum in the potential field. This surfaces in artificial potential fields in the form of local minima, in which robots can get stuck. Several techniques exist to allievate this issue, most of them are probabilistic and rely on the robot detecting when it's stuck. [Bell and Weir, 2004] has a discussion of the different types of local minima and how to avoid some of them. [Bell, 2005, chap. 4] has a discussion of which types of obstacle shapes that cause local minimum problems. [Bell, 2005, chap. 5] has a lengthy discussion of the various techniques that exist to alleviate the local minima problem.

An even more difficult problem to solve is when a system oscillates. This is not uncommon in real particle systems and it is much harder for a robot to detect than simply realizing that it is immobile and not at its target.

[Rimon and Koditschek, 1992] presents an approach for creating potential functions that do not have the problem of local minima. These functions, named navigation functions, are provably correct and guarantee collision-free motion and convergence to the destination of the robot. This technique is more computationally expensive and it does not easily generalize to situtations in which the robot's target or the obstacles in the environment are mobile. Furthermore, restrictions are placed on the shape of obstacles that makes this method impractical in many real domains.

[Koren and Borenstein, 1991] notes the following problems as present with traditional potential fields methods.

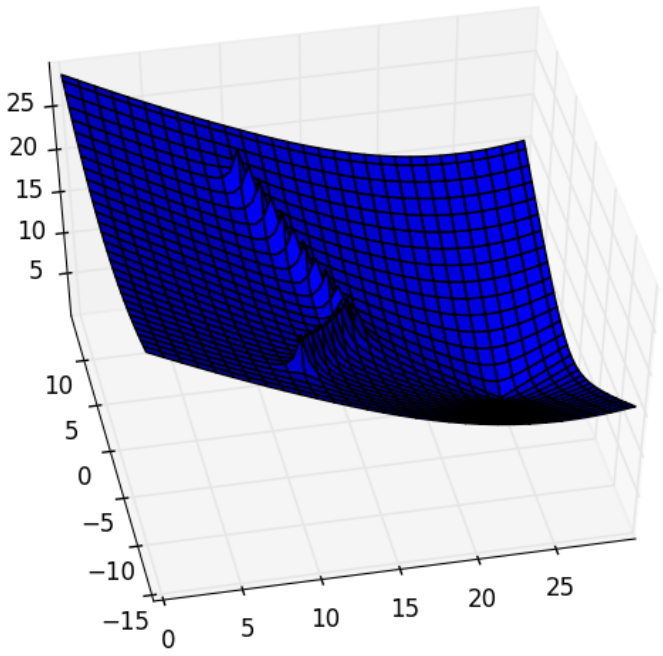

Figure 2.3: The horse shoe shape or v-shape of obstacles creates a local minimum.

- Local minima in which robots can get stuck.
- Robots are unable to enter in the passage between closely spaced obstacles.
- Oscillations near obstacles.
- Oscillations in passages between closely spaced obstacles.
- Goals can be nonreachable with obstacles nearby.

There are various ways of solving these problems, but in many cases they are impractical or pose restrictions on the robots domain that make them unusable. [Rimon and Koditschek, 1992] requires that all objects are elliptical or star-shaped, but the method presented therein can provably handle the above problems. [Ge and Cui, 2000] solves the last problem and it is possible to combine this method with local minima avoidance heuristics and techniques.

Artificial potential fields for distributed behaviour

Inspired by [Reynolds, 1987], there have been many ventures into using artificial potentials to create distributed behaviour among simulated creatures and robots. This paper presents an interesting result of simulated bird flocks. Each bird here is a very simple actor, but the flock as a whole can exhibit complex and interesting behaviours. This approach is very similar to artificial potential fields found in robotics, but instead of obstacle avoidance the focus is creating complex flocking behaviour. Each simulated bird has simple rules for how to react to its local environment, analogous to force laws or potentials from other research.

[Reif and Wang, 1999] takes this one step further and use potentials to create interesting and complex behaviours in very large systems of robots through simulation. Two different types of approaches are shown here, *force laws* are shown to be highly efficient for creating certain types of distributions and behaviours among the robots and *spring laws* are shown to be able to create exact structures in their relations. These structures are named spring laws because they are rigid, so long as there is room for them to be, but they are elastic when the group of robots need to be compressed to avoid obstacles. They do however maintain the structure of the group while compressed, only the distances between members of the group are adjusted.

Simple building blocks consisting of two pre-selected constants c and σ are used to build force laws between robots. Given the vector $\vec{r_{ij}}$ between two robots i and j, force law between them is a sum of terms built from different values for c and σ as in (2.11).

$$f(c,\sigma) = \frac{c}{|\vec{r_{ij}}|^{\sigma}} \frac{\vec{r_{ij}}}{|\vec{r_{ij}}|} = \frac{c \cdot \vec{r_{ij}}}{|\vec{r_{ij}}|^{\sigma+1}} \tag{2.11}$$

This chooses some magnitude for the force between i and j and multiplies it by the unit vector between them to obtain the direction of the force. By setting c negative or positive, $f(c,\sigma)$ can be used either as a repulsive potential function or an attractive potential function. Because the authors focus on group behaviour, most objects are both repulsers and attractors. This is done by setting the potential between two objects to be (2.12) where c_1 is negative and c_2 is positive and plugging into (2.11).

$$\mathbf{F}(i,j) = f(c_1,\sigma_1) + f(c_2,\sigma_2) \tag{2.12}$$

It is shown that for robots governed by such force laws, there exists some equilibrium distance d. The significance of the equilibrium distance is that this is where the force applied from (2.12) is 0, such that robots would be at rest. The

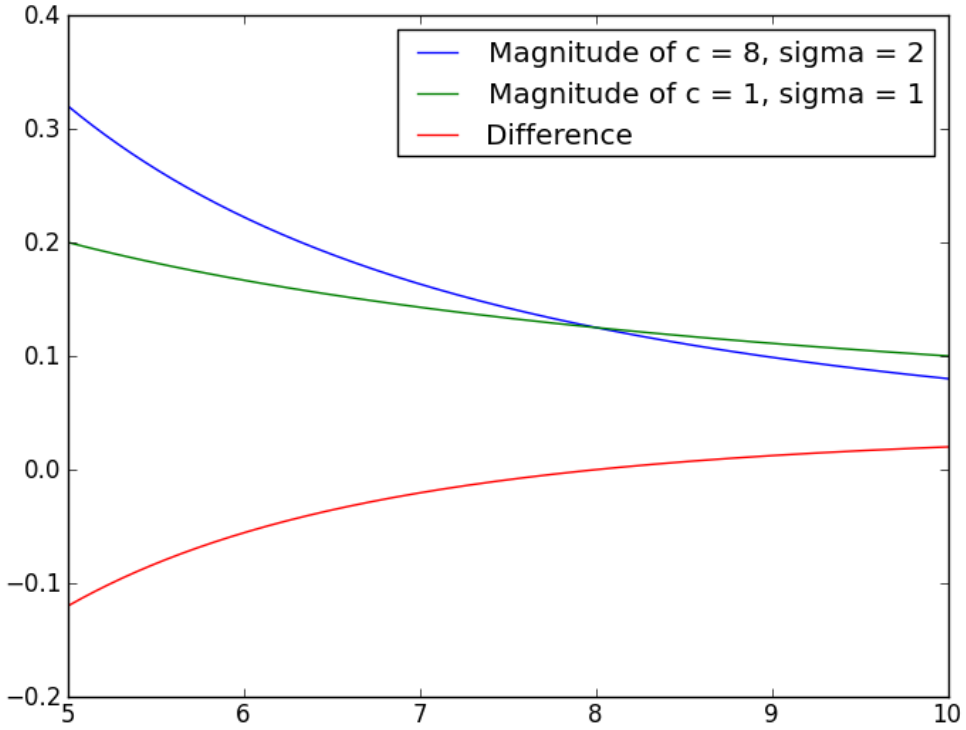

Figure 2.4: Social potentials for $c_1 = 1, \sigma_1 = 1, c_2 = 8, \sigma_2 = 2$ and their difference.

authors use the concept of equilibrium distance to calculate the expected density of robots in a group that all obey the same force law.

Forces are used to set velocities directly, such that the velocity of a robot i in an environment with robots 0 through n obeying the same force law **F** becomes:

$$\vec{v_i} = \sum_{j<n, j\neq i} \mathbf{F}(i,j) \tag{2.13}$$

The authors show how to define force laws such that they can generate particular types of desirable behaviours.

In Figure 2.5 a social potential is used to calculate the vector field a robot would see. In this case, there is one obstacle, located in $0, 0$ and a goal, located in $15, 15$. The force law for the obstacle is a simple repulser with $c = 10, \sigma = 2$ and the force law for the goal is a simple attractor with $c = -10, \sigma = 1$. The plots have been scaled, as the forces very near the obstacle and the goal approach positive and negative infinity respectively. Figure 2.6 shows the same field plottet on a 3d

surface. The path of the robot in this plot would be equivalent to that of a ball that was dropped onto this surface.

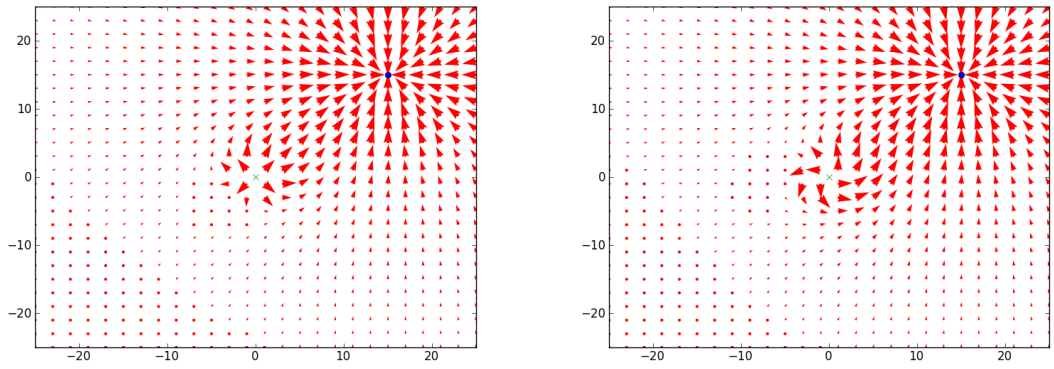

Figure 2.5: Vector field from a simple social potential force law and a tangential potential field that resolves the local minima issue.

There is a local minimum along the line that goes through both the goal and the obstacle in this field. In this case, we could add a tangential potential field around the obstacle to circumvent the problem. Tangential potential fields have a rotation around their center, such that a local minimum is avoided. This can be done by adding a fraction of the normal of the vector from the robot to the obstacle to the potential, see Figure 2.5.

The system developed by the authors features groups of robots, where groups are allowed to have separate internal and external force laws. This means that robots in the group affect each other by the same rule, but that they affect robots in different groups with a different rule. Using these simple building blocks, the authors show a variety of different behaviours, such as escorting, searching minefields or moving formations to be possible. One of the simplest behaviours is the clustering force law, an example of an initial and later configuration of robots following this force law can be seen in Figure 2.7. Between each pair of robots here is a force law with $c_1 = 15, \sigma_1 = 2, c_2 = -1, \sigma_2 = 1$.

Force laws need not be reflective, that is a robot can exist that pulls and repulses other robots without being affected by them.This gives rise to an interesting idea of having a leading robot that others follow in some formation. This robot does not need to be a real robot, [Leonard and Fiorelli, 2001] shows that virtual robots can be accounted for to adjust behaviour. These may even pop in and out of existance while simulations are running to provide leadership for formations or allow for change in formations. We note in passing that using some layouts for virtual leaders can be used to force groups of robots into different types of formations.

[Leonard and Fiorelli, 2001] also show several techniques for developing groups

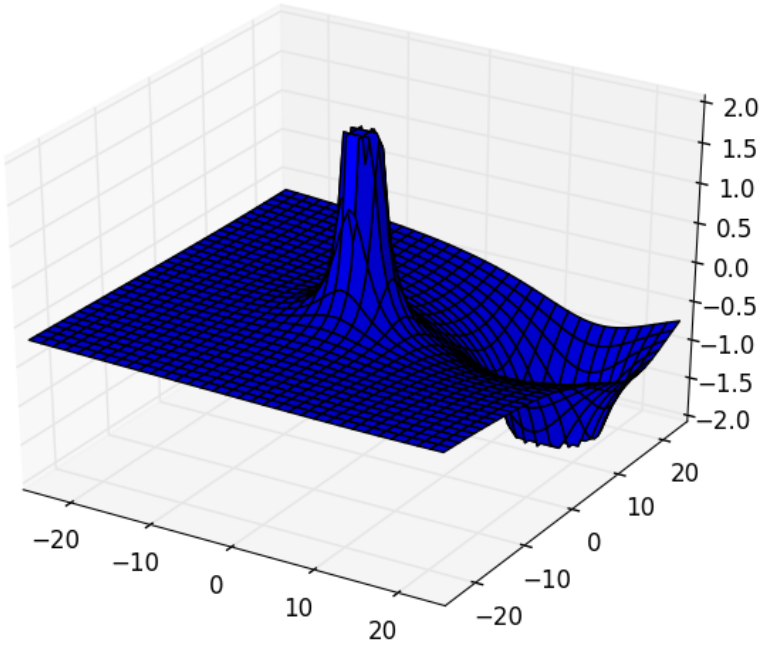

Figure 2.6: Vector field from a simple potential force law in 3d.

of robots that obey a certain structure or exhibit a certain behaviour. In the framework developed herein, *local* potential fields are allowed - that is, fields in which invidiuals are only affected by their nearest neighbours (and possibly virtual leaders).

Combining artificial potential fields with other techniques

[Vadakkepat et al., 2000] shows promise in applying an multi-objective evolutionary algorithm to choose weights on simple potential functions of the form (2.14), where r_g is the distance from the robot to the goal and r_o is the distance from the robot to obstacle o. Here, a and n are constants that are evolved. The robot has a constant velocity and uses the force only to set its angle of movement.

$$\mathbf{F} = \frac{1}{r_g} + \sum_o \frac{1}{\left(ar_o\right)^n} \tag{2.14}$$

An additional equation is introduced for local minima avoidance. When the robot detects that it is at a stand-still (when (2.14) is 0), it applies an escape force.

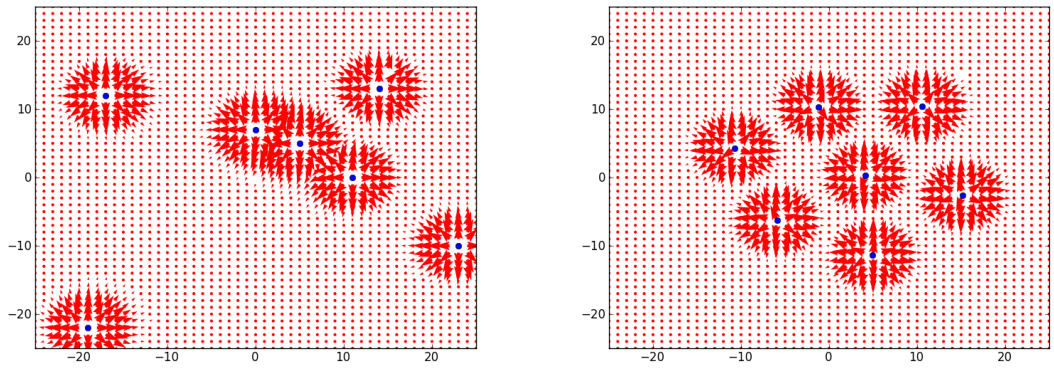

Figure 2.7: Robots sharing a social potential force law. Initial and later clustering and vector field.

The escape force has additional parameters b, c, d and m that are also evolved. The authors observe that their method produces optimal smooth paths for cases where the environment contains moving goals and targets.

This method is carried over to [Vadakkepat et al., 2001] where it is tested with a robot soccer system. The robots are able to seek the ball while maintaining collision free paths. Although the robot is able to kick the ball and maintains relatively smooth paths, the authors conclude that the potential field functions need to be optimized in real-time to achieve good results.

Using the idea of virtual leaders, it is possible to combine classical pathfinding techniques such as A*-search with artificial potential fields with relative simplicity. To do this, one would let a virtual leader follow a path discovered by A*-search and let all other robots be attracted to it. This enables the usage of potential fields for group-interaction and a different technique for pathfinding, making the method very versatile. Even using A*-search for pathfinding, it is still possible to use evolutionary techniques to evolve the weights for the potential field functions that govern the group internally.

Artificial potential fields in real-time strategy games

[Hagelbäck and Johansson, 2008] [Hagelbäck and Johansson, 2009] show some promise in applying potential fields to the domain of real-time strategy games. A computer program that plays a strategy game implemented on the ORTS platform is documented. This program won the 2008 ORTS tournament. The Berkely Overmind project impressively won the 2010 AIIDE competition, using trained potential fields as a very important part of its technology. The Overmind uses reinforcement learning to train its potential fields [Klein, 2012].

Notes on complexity and relation to N-body problem

Artificial potential fields are very closely related to the N-body problem - solving them exactly can be shown to equivalent. In practice, they are instead solved similarly to how N-body simulations are created. This means that the barnes-hut method [Barnes and Hut, 1986] can be adopted to solve them efficiently, opening up for the possibility of highly complex environments for very large systems of robots that are controlled with artificial potential fields. Using Euler's method for simulating artificial potential fields is sufficient for having several hundred objects that affect each other with force laws on modern computers and run them in real-time according to [Reif and Wang, 1999].

2.3 Optimization techniques and evolutionary computation

Local search and optimization algorithms

Several different techniques are applicable to locate good weights for potential functions. Because potential functions map from reals to reals, many search algorithms are hard to adapt to discovering good weights. This is a review of some techniques that can work well together with contineous, real-valued search spaces.

Hill-climbing search

Hill-climbing is a very simple search algorithm that will find a local maximum. It achieves this by simply moving the search state in the direction that causes the greatest increase in the evaluation of the search state. Hill-climbing can not guarantee an optimal solution and it can get stuck on plateaus that border local maxima if there are restrictions in place for how many times it can move the search state with no improvement in its evaluation. There are many variants of hill-climbing, one of the most novel ones being the *random restart* version. This algorithm works by attempting to restart the search at another, random state if it fails to find a maximum that is good enough or one that is known not to be the global maximum. For some problems, it is very efficient ([Russell and Norvig, 2003]). A trivial modification to this that takes advantage of modern computer architectures is to run hill-climbing searches in parallell.

Simulated annealing search

Simulated annealing ([Kirkpatrick et al., 1983] [Russell and Norvig, 2003])refines on hill-climbing search significantly. Instead of always moving to the best successor-state, it picks a random successor-state. If the new state is better, the search proceeds from there. If it is not, it proceeds from that state with some probability $p = P(S, S', T) < 1$. P scales downwards with the loss of quality from S to S' and scales upwards with the temperature T. As the search progresses, T decreases and when it reaches $T = 0$, it becomes a special case of simple hill-climbing search. Simulated annealing is a good combination of random walking and greedy search and has been shown to be very efficient for a multitude of problems. [Floreano and Mattiussi, 2008] lists simulated annealing search together with evolutionary computation techniques because it so closely resembles them. It is very similar to evolutionary programming, using a special simulated annealing selection strategy. Simulated annealing can be used with vectors of real by letting the random search state be generated by randomly changing an element in the vector.

Particle swarm optimization

Particle swarm optimization is a machine learning technique that is driven by cooperation between the involved particles. This is different from evolutionary techniques, which we shall later see are driven by competition between the involved individuals. It is loosely based on cooperation in bird flocks in search of food ([Kennedy and Eberhart, 1995]).

A PSO instance consists of a number of particles that move on the search space, looking for a global optimum. A particle knows its location in the search space and its performance. Initially, the search will have the particles distributed randomly in the search space. The particles communicate their performance to their neighbours and shift their positions by adding up a fraction of their previous direction (inertia), a fraction of the direction to the position where it found the highest performance so far (cognitive rate) and a fraction of the direction towards its highest performing neighbour (social rate).

To avoid stagnation and to enable the optimization to explore novel search areas, a degree of uncertainty is required. This is usually achieved by making the particles unable to know locations exactly, so that the location of the best recorded performance and the location of the best performing neighbour has some uncertainty.

[Floreano and Mattiussi, 2008] notes that PSO performs well on optimization problems with real-value functions compared to other techniques.

Evolutionary computation techniques

Evolutionary computation techniques present the textbook classic of how inspiration for computational processes can be found in nature. The idea is very simple - to apply the process of evolution on data. A population of individuals that directly or indirectly encode solutions to some search problem are generated. This is the initial generation. In every generation, the currently existing individuals are evaluated with a fitness function that is problem-specific. A group of individuals are then selected from the current population to be used for creating the next generation. These may be combined and possibly mutated before they are placed in the new generation.

While these processes are inspired by nature, they are fundamentally different in several ways [Floreano and Mattiussi, 2008]. Namely, the fitness function is created by humans to direct evolution in an evolutionary computation. Real evolution is not directed, it is completely open-ended. All individuals created by an evolutionary computation will tend to try to satisfy some fitness function - this fitness function defines the chances of their genes to be passed onto successive

generations. In nature, we could say that the fitness of an individual is the success it had in imparting its genes - it is not possible to create a real-valued function to order individuals in other ways.

Genetic Algorithms

A genetic algorithm is a very well-known case of an evolutionary computation technique. In this case, the population consists of strings from a finite alphabet, usually the binary alphabet $\{0, 1\}$. Evolution takes place by repeatedly applying genetic operators to the population to produce new generations. It halts after some amount of generations, or when a predefined predicate for fitness is true. The following types of operators are commonly found in GA systems:

Crossover
: Combines the genes of two individuals into new individuals. Crossover attempts to preserve good genes while still exploring the search space.

Mutation
: Performs a random alteration of an individual. A common mutation is to flip a bit in a genotype that is represented by a bitstring.

Selection
: Typically a weighted random process of selecting which individuals that will be allowed to fill the next generation.

Within the framework defined by these, there is a huge amount of variations that can be defined [Floreano and Mattiussi, 2008]. Although the mechanics of evolutionary computations that use different representations of individuals may be similar to the canonical GA, they are typically not named as such.

Selection of evolutionary computation operators

For the problem domain in this thesis, working with vectors of real numbers is more important than working with bitstrings. A short discussion of operators that work well with this representation follows.

Arithmetic crossover
: Produces one genome from two parents, by taking the average of n randomly chosen positions of the parents' genomes.

Blend crossover
: Similar to arithmetic crossover with added mutation(s). Values are chosen

randomly from the range bounded by the parents' genes, but this range is extended by some given amount.

Uniform crossover
Produces two genomes from two parents, by swapping genes at n randomly chosen positions of the parents' genomes.

Gaussian mutation
For each gene in the genome, has a chance of adding a value randomly chosen from the gaussian distribution with a given mean and variance.

Rank-based selection
Ranks all individuals from best to worst and selects individuals with probability proportional to its rank.

Tournament selection
Picks some number of individuals from the population at random and choose the individual with the highest fitness. To select n individuals, n tournaments will be run.

Generational replacement
Replace entire population by their offspring. Sometimes comes with *elitism*, allowing some of the fittest individuals from the old population to survive to the next generation.

Plus replacement
Replace the entire existing population by the best population-many elements from the combined set of parents and offspring.

Evolutionary Programming and Evolutionary Strategies

Evolutionary programming is more commonly used with vectors of reals as genotypes and sometimes operate directly on phenotypes. It generally applies perturbations with a method similar to gaussian mutation (see 2.3) and very often it uses tournament selection with gradual population replacement. EP does not use crossover.

Evolutionary strategies are similar, but the variance of the distribution used for mutation is genetically encoded and evolves alongside the rest of the genetic material in individuals.

Chapter 3

Implementation and Experimental Setup

3.1 Requirements

To assist in the process of implementing a system to allow for the experiments in this thesis to be done, a small list of requirements were developed. These provide guidelines for design decisions that needed to be done in the implementation.

Flexibility
The simulation system must be flexible enough to support several types of potential fields, actuators and some degree of scripting.

Visualization
The simulation system must allow for visualization of potential fields, objects manoveuring in these and their obstacles. The system should support doing this in real-time to allow for rapid experimentation of potential field setups.

Extensibility
The system must be able to be extended to allow for integration of third-party libraries that perform optimization through the use of genetic algorithms and similar techniques.

Correctness
The system must be able to reproduce previous results in the field - in particular, the results presented in [Reif and Wang, 1999] are important for this thesis.

Ease of use
The system should be relatively easy to use, as many experiments will be performed. This means that it is unacceptable to spend a lot of time setting up simulations when the system can assist the process to make it go quicker.

Efficiency
To ensure that it is possible to combine the system with optimization techniques, it is necessary that simulations that produce minimal output can be run very quickly. If this criterion is not met, experimenting with several different optimization techniques will take too long.

These requirements affected the implementation in various ways. To ensure that the system was flexible and easy to use, Python was chosen as the programming language to implement it in. Python is a mature language that easily allows for scripting and has a large body of mature modules for visualization and optimization techniques. While it is a byte-compiled language that is not particularly efficient for numerical code, it is easy to interface to C, such that inefficient code can easily be rewritten to C when necessary.

3.2 Implementation

Dependencies

The simulation system that was developed is mainly a library that the user can call from python programs. Additionally, a small core of scripts follow that produce or consume JSON-data for purposes such as running simulations, plotting or generating animated videos for visualization. A small number of third-party libraries were used to build the system, these were:

Cython
Cython provides a compiler that will generate highly efficient C code from Python code. It does this by extending the Python syntax to allow for type specifiers. This was used for the most numerically intensive code in the system.

Matplotlib and numpy
Matplotlib is a library for producing pretty plots of many types, including 3d-surface plots, contour plots, scatter plots and vector fields. numpy is a requirement for Matplotlib and provides highly efficient code for array manipulations.

inspyred
inspyred is a framework that provides several interesting optimization techniques, such as genetic algorithms, simulated annealing, hill-climbing search and particle swarm optimization.

pygame
pygame is a set of bindings to the SDL library and is used to produce streaming visualization of running simulations.

ffmpeg
: ffmpeg can combine several pictures to generate an animation in a well-known movie-format.

Design and implementation

The design for the system is relatively simple. Starting at the bottom, the units that are simulated are instances of the class SimObject. This class can represent pointmasses or line segments. An object of this class has an instance of SimObjectData, in which forces and velocity are stored. Arbitrary data can be attached to a SimObjectData instance by adding them to a hash-table. SimObjects support attaching hooks to be run before, during or after the simulation step such that actuators can be defined by client code and constraints such as maximum speeds can be enforced. Additionally, adding hooks here allows client code to detect and get out of local minima. These two classes were compiled to C with the help of Cython, as was the vector calculation code they use.

Also written in Cython are the PotentialTerm and ForceLaw classes. A ForceLaw is a container and convenience class for PotentialTerms - calculating the force from a ForceLaw will merely sum up the forces from all its terms. A ForceLaw calculates the force that a SimObject would exert at a given point, using that ForceLaw.

A SimObjectGroup is a container for several SimObjects that naturally belong together. In addition to its constituent objects, it contains an internal ForceLaw and an external ForceLaw. The internal ForceLaw is applied between objects in this group, whereas the external ForceLaw is used to allow the objects in this group to affect objects in other groups.

The heart of a Simulation instance is the GroupManager class. This is a container for several SimObjectGroups. It contains a list of relations and knows which groups that should affect each other, such that for example groups of obstacles will provide a push to all groups that contain units that have obstacle-avoidance.

A Simulation instance provides the machinery necessary to update all of the aforementioned classes and can be read in from configuration files. It is also the hook-point for live visualization, offline plotting, generating data dumps and doing performance measurements. All classes that have been mentioned so far can be serialized and deserialized to JSON.

The system provides one main command, apfmain.py, which is used to manage project folders consisting of plot-data, generating movies and running simulations. A script is also generated inside these project folders to allow for more fine-grained control of simulations than what can be done with simple data files. This is needed, for example, to have SimObjects that use different rules for move-

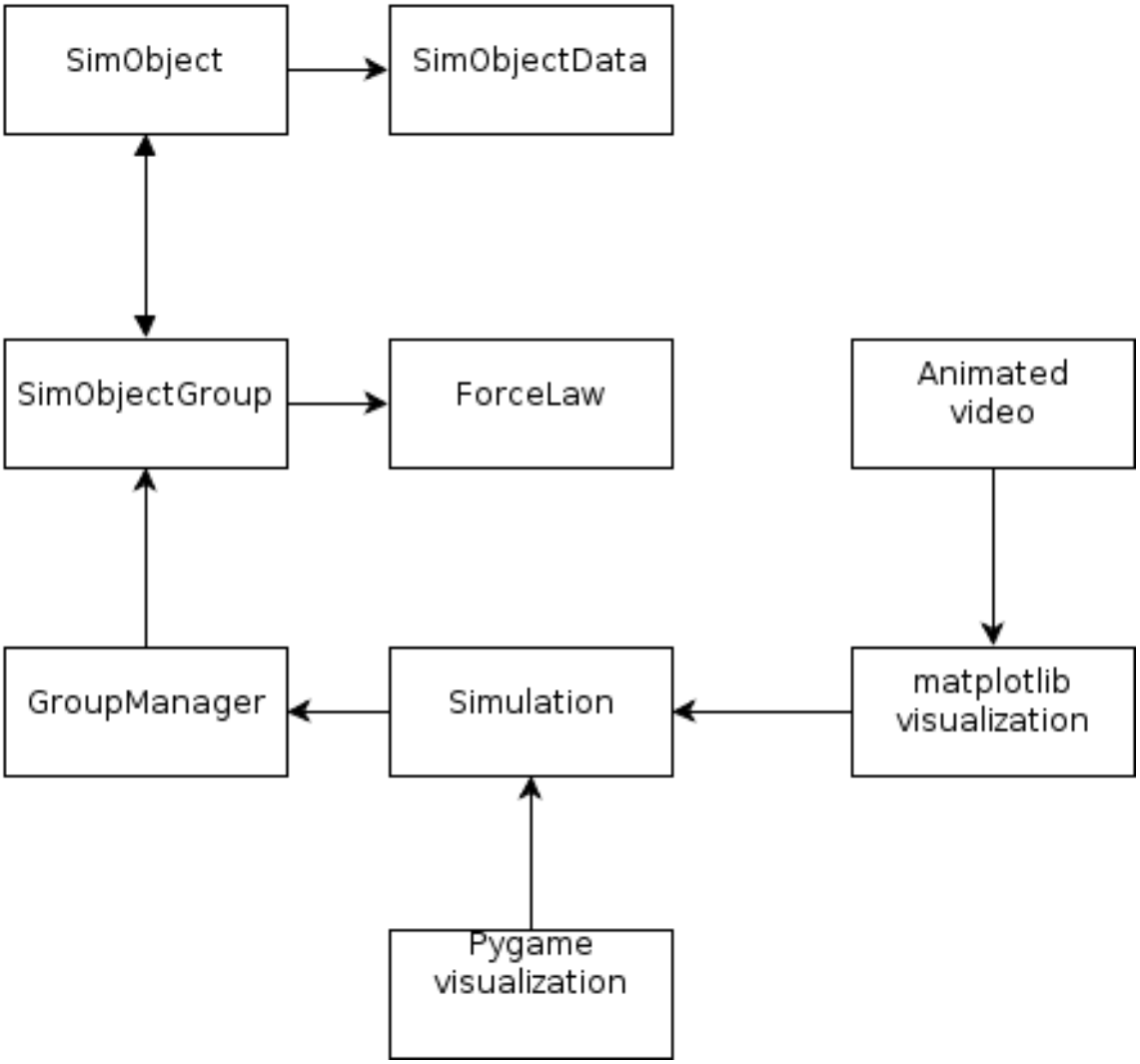

Figure 3.1: High level look at simulation architecture.

ment than potential fields (for example hardcoded patrol paths). By default, the script that is included will visualize the running simulation in a pygame window.

Notes on SimObjects

SimObjects can be used either to define straight lines or points. When potential functions are calculated between SimObjects, the shortes distance between them is used to calculate the force, in general. However, in some cases it might be beneficial to have lines that exert forces only along its perpendicular line. For this reason, two different types of distance calculations are implemented. One of these is only applicable for lines with “round edges”. These are lines that act is if their end points are point-masses, such that there is a circular potential field around their end points. Lines without this property do not exert any forces to points that

are not above or below the line-segment.

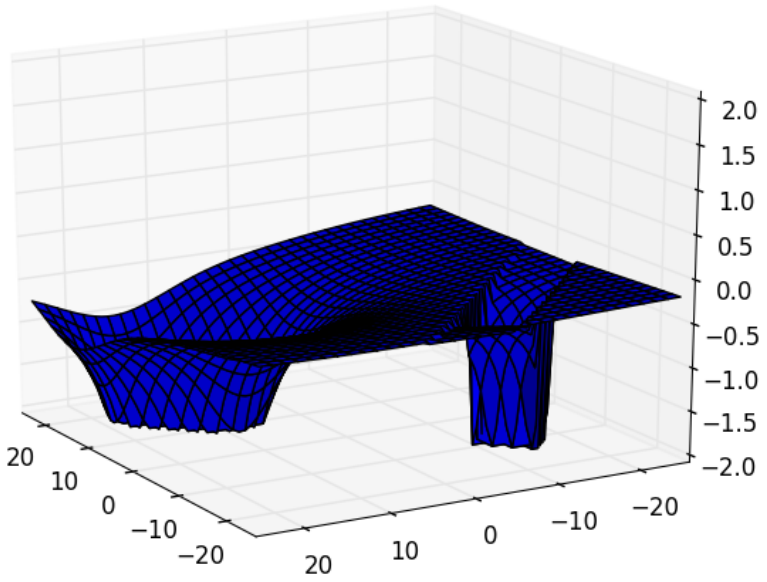

Figure 3.2: An attractive line with “round edges” and one without.

A SimObject can be queried for properties. If it fails to find the property that is being asked of it, it will fall back to asking its group for a value for it. Arbitrary properties can be attached to a SimObject by adding it to the hash-table located in the SimDataObject. Similarly, SimObjectGroups have a hash-table called data that serves as another location for client-provided data to reside. Only three properties have special significance to the simulation system, the “plt” property is a matplotlib colorspec that defines how the object looks if plotted and “max_speed” is the maximum amount of distance an object is allowed to move in one time unit. There is also a “draw” property that is useful for persistent objects that have a utility purpose, but take no actual part in experiments. This is here to let client code take advantage of distance-vector calculation code implemented in SimObjects without littering plots with objects that do not have any forces associated with them.

Performance analysis

To discover whether the technology of potential fields is usable for AI in computer games, it is necessary to have, at the very least a rudimentary understanding of

how computationally expensive they are. A simple benchmark of the simulation system was conducted to test whether it is at all applicable for modern computer games. It is not necessary to calculate the potential fields for all objects at each frame update, although this will lead to better results. We expect the calculation times to increase quadratically with the amount of objects in the simulation, as the algorithm used is a very basic adaption of Euler's method of integration.

The performance test is simple. For each input size, 3 simulations were run. The input size determines the amount of objects in the simulation. They are randomly distributed in a large area. Each simulation is run for 1000 update steps and the time for each update step is averaged. The result is the average update time for the 3 simulation runs and we calculate the standard error of this as well. All the objects are added to the same group and this group uses an internal force law. If $r(i,j)$ is the distance between units i and j and $\vec{u}(i,j)$ is the unit vector that gives the direction, then the force law between them is (3.1).

$$f(i,j) = \vec{u}(i,j)\Big(\frac{-1}{r(i,j)} + \frac{20}{r^2(i,j)}\Big) \tag{3.1}$$

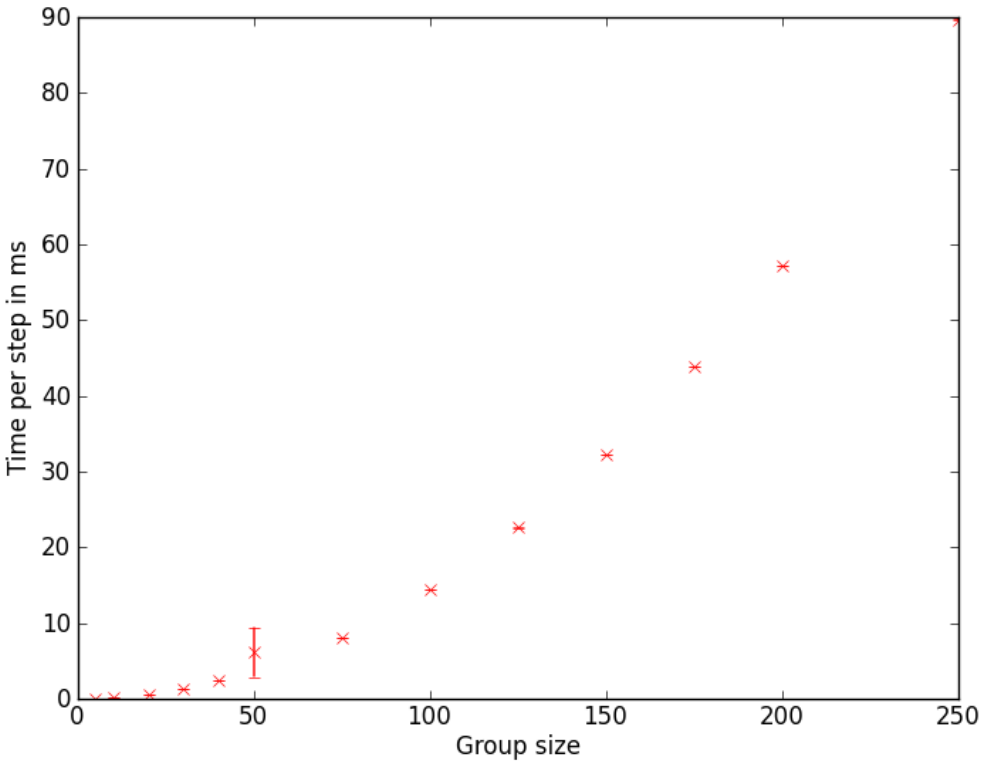

Figure 3.3: Runtime per update step for different simulation sizes, with errorbars.

To make it actually viable to use potential fields for artificial intelligence in games, it needs to be possible to update the forces on units several times per second with large groups. Although the implemented simulation system is not heavily optimized, it does give an indication of the cost of running artificial potential fields. At the very least, a commercially developed game using this technology would not do worse. Figure 3.3 documents the time taken to run one simulation step, in milliseconds, for several different input sizes. The benchmark was conducted with a commodity laptop, sporting an Intel(R) Core(TM) i5 CPU running at 2.40GHz and one core was used. As expected, the runtime seems to scale quadratically with input size. With the simulation system written in Python, a byte-compiled dynamically typed language and most games written in compiled, statically typed language, one would expect a very significant speed-up in a commercial implementation.

It is also not necessary to sample the forces on the units at every frame update, although this improves precision of movement. We can conclude from this that the efficiency of the method is sufficient for implementation as a subsystem of a commercial game.

3.3 Experimental setup

Each experiment is a set of data files that are consumed as input by a simulation, which is either run as default simulation or a script that enhances the simulation in some way. Enhancements can either be additional data that is added to the simulation programmatically or it can be code.

Simulations produce plots, movie and machine readable data files as output. An experiment is described by the following:

Objective
: The stated objective of the simulation - that is, the desired behaviour of the simulation.

Environment
: The environment in which the simulation is run. A description of the general rules of physics and any obstacles present.

Motivation
: Why the experiment is useful - how the desired behaviour of the simulation could be useful for a game-ai agent.

Performance Measure
: How the performance of the simulation is measured. This is a numerical measurement that is either being maximized or minimized, such that sim-

ulation instances can be ranked according to how close they are to desired behaviour.

Expected Outcome
A short description of the characteristics of the expected results.

Physical quantities in the simulation system

While the simulation system does not attach any sort of physical measurement unit to any quantity it deals with in a simulation, it is useful to establish what they correspond to. For example, it is useful to discuss the simulated time elapsed in terms of seconds, even though there doesn't need to be a 1-to-1 correspondence between simulated seconds and real seconds.

Simulated time is expressed as a sum of time-deltas, as is common for methods based on Euler's method of integration. That is, a time-delta for each step of the simulation is selected before it is running and then that is used to step the simulation until it is done. Therefore the simulated time is the amount of steps multiplied by the time-delta at each step. For finer grained simulations, the time-delta can be reduced. In most cases, a value that corresponds to an actual framerate is a pretty good value. By using a time-delta of $\frac{1}{24}$, movies produced by the system will have a 1-to-1 mapping between real-time and simulated time, if every simulation step is plotted.

Distance has no unit of measurement attached to it either and neither does velocity or acceleration. The normal relationships between these hold. Velocity is distance travelled per time unit, acceleration is the difference in velocity per time unit. Because there is no actual correspondence between real-world units (such as pixels) and the simulation space, plots produced by the system have axes, labeled with coordinates. Arrows are scaled liberally in general - the relationship that objects with long arrows have larger forces than objects with short arrows holds true because scaling is uniform.

The simulation system does not incorporate a concept of mass, drag or friction. These concepts could easily be added through the use of the simulated objects hash-table and usage of actuator-hooks.

Minimal experiment

A minimal experiment contains two json-formatted files. One of these correspond to settings that should be used by the simulation system (such as what names to give output files, time-delta to use) and the groups of objects that should be simulated. The time-step (this is called "dt" in apfrc, the setting file), and the amount of simulation "steps" to run need to be defined.

Additionally, there should be one group of units and an internal force law between them in the input file. By running the system from its *apfmain.py* command with the parameters *create <project-name>*, such a minimal experiment will be created in the folder *project-name*.

Notes on the use of inspyred

The optimization techniques used in the experiments all come from the python library inspyred. It implements all the techniques outlined in 2.3 and more. In inspyred, all optimization techniques are instances of an evolutionary computation baseclass, which enables them to be used in almost exactly the same way. For each optimization technique, the following must be supplied

- Either an initial population or a generator function that can generate individuals on demand.
- A fitness function that evaluates the fitness of an individual.
- A bounder that encloses the search space for the specific genes in an individual.
- A termination condition.
- A population size.

Unless otherwise stated in the experiments, all the default values set by inspyred are applied to optimize the potential field weights.

Genetic Algorithm for vectors of reals

An evolutionary computation built nearly like a canonical GA. This uses rank selection and generational replacement. To adapt it to the vector of reals implementation, it uses blend crossover. The default parameters are:

- crossover_rate = 1
- mutation_rate = 0.1
- gaussian_mean = 0
- gaussian_stdev = 1

Evolutionary Strategy

This uses plus replacement and gaussian mutation. The parameters for the gaussian mutation are evolved using an adaptive mutation specified by the inspyred system. The default values for parameters are:

- $\tau = \frac{1}{\sqrt{2\sqrt{n}}}$, where n is the length of a candidate.

- $\tau' = \frac{1}{\sqrt{2n}}$
- $\epsilon = 0.000001$ is the minimum strategy parameter.

The parameters for the gaussian mutation are updated as:

$$\sigma_i' = \sigma_i + e^{\tau \cdot N(0,1) + \tau' \cdot N(0,1)} \tag{3.2}$$

$$\sigma_i' = max(\sigma_i', \epsilon) \tag{3.3}$$

Simulated Annealing

This uses a special replacement strategy called simulated annealing replacement. All individuals are parents and parents are replaced by their offspring depending on the simulated annealing evaluation. Offspring are gaussian mutated variants of their parents. If temperature and cooling rate are not provided, the computation will use the range $(1, 0)$ as a cooling schedule, but this works only when the computation knows either the maximum number of generations or the maximum number of fitness evalutions it can perform.

- mutation_rate = 0.1
- gaussian_mean = 0
- gaussian_stdev = 1

Particle Swarm Optimization

Assumes sequence of reals and uses distance from previous timestep to calculate the velocity of the individuals in the population. This PSO algorithm is based on [Deb and Padhye, 2010]. It uses the following default values:

- Topology is set to a star topology.
- inertia = 0.5
- cognitive_rate = 2.1
- social_rate = 2.1

Genotypes and bounds used with inspyred

Each experiment provides inspyred with genotypes that are represented as vectors of real values. Each gene in the genotype is bounded by an experiment-specific bounding function. A gene has a lower and an upper bound and if it is outside this range, it is simply moved inside it, according to this simple procedure:

```
def bound(genome, lower_bounds, upper_bounds):
    for each index, gene in genome:
        if gene < lower_bounds[index]:
            gene = lower_bounds[index]
        if gene > upper_bounds[index]:
            gene = upper_bounds[index]
```

To provide a good seed population, care needs to be taken while generating it. If the population is generated entirely randomly, it will most likely consist entirely of individuals with genes that need bounding. To prevent that from happening, the generator is also aware of the bounds. Two different methods immedately come to mind - generating each gene as a uniform random distribution between the lower and upper bound, or using a gaussian distribution with the average of the lower and upper bound as a mean and a standard deviation that guarantees that most generated genomes will require no bounding. The second method is interesting because it will spread the genomes in the more interesting area of the search space - presumably boundary values are not so interesting. This enables the experiment to "hint" at where the good values for genes are, which is a very favourable property to ensure good results. Therefore, this method was used:

```
def fill(genome, lower_bounds, upper_bounds):
    for each index, gene in genome:
        low, high = lower_bounds[i], upper_bounds[i]
        mean = (low + high) / 2
        stddev = (high − low) / 5
        gene = rand_gauss(mean, stddev)
```

Although this can generate genomes that will have bounded genes, it will take place extremely rarely because of the low deviation of the distribution used. The initial population will however explore a smaller area of the bounded search space, such that this requires the bounds to be set with some care.

Experiments define their own evaluation functions, but these all work by the same principle - they accept a genome, create a new artificial potential field simulation and apply the genome to its weights. They then return the score of this simulation as the fitness of the genome. Typically, each gene in a genome sets one parameter for a particular potential function. For example, for social potentials (see (2.12)), a gene of $[-1, 1, 5, 2]$ could correspond to the potential function with $c_{\text{att}} = -1, \sigma_{\text{att}} = 1$ and $c_{\text{rep}} = 5, \sigma_{\text{rep}} = 2$. The evaluation functions encode this information, they assign the parameters according to their own scheme.

Chapter 4

Experimental Results

4.1 Formation experiments

Marching in a line

Objective

Create a simulation of units that march in a flat formation. The units are initially randomly spread in a small area and need to converge to a line formation quickly. The line formation should move a short distance in a coherent formation.

Environment

The environment does not have any obstacles. Units can move at a limited speed. Their velocity is calculated by adding acceleration to it at every timestep. Their acceleration is calculated from potential fields. Units are not allowed to collide.

Motivation

A line formation provides a benchmark for a relatively simple manouver that an rts-agent needs to be able to perform. It is commonly used to block passage for enemy units. It is also applicable for sweeping large sections of terrain, looking for hidden objects, such as mines.

Performance Measure

Minimize the amount of units that collide and the average distance from units to the line that is the center of the formation.

Expected Outcome

A setup of potential functions that guides units into a line formation. We expect a tighter line formation with less space between the units and the line from applying optimization techniques.

Description

This experiment takes place in a 2-dimensional world, using a contineous coordinate system. The extent of the world is not bounded, but the units are initially distributed in a 60 by 60 box, centered around the origin and they are expected to stay within this area. To start with, positions for 10 units within this box are generated, using a random uniform distribution inside the box (with the constraint that no unit is placed within a distance of 5 from another).

Units are assumed to be circular, with a radius of 1.25. The distance between units is calculated from their centers, which means that two units are colliding if their distance is ≤ 2.5. If two units collide, they will both be removed from the simulation. Units are actuated by calculating their acceleration vector and adding it to their current velocity vector. Their acceleration is simply the sum of forces on them. Units are allowed to move at a maximum velocity of 0.5 per simulated second.

The line that marks the center of the formation is initially positioned such that its center is at the location $(-25, 0)$. Its extent is 44 and it is oriented along the y-axis. It will stay in this position until 25 simulated seconds have passed, after which it will move in a straight line to $(25, 15)$. Its speed is limited to 0.3 per simulated second. Once there, it will rotate counterclockwise exactly twice. The line rotates with an angular velocity of 0.0375 radians per simulated second, which means that it moves faster at its endpoints than units are allowed to. After the line has rotated twice, the simulation terminates. The simulation will be run with a timestep of 0.2.

Performance measure

Let t denote the current timestep, $n(t)$ the amount of units alive at timestep t and $r(i, t)$ the shortest distance between the center of unit i and the line at time t. Then the performance of the simulation at timestep t is given as:

$$p(t) = \frac{1}{n(t)} \sum_{i<n(t)} r(i, t) \tag{4.1}$$

The performance of the simulation is the average performance of each timestep and a penalty for the amount of units that died. Let $T(S)$ be the amount of timesteps in the simulation S and $d(S)$ the amount of units that collided. Then, the performance of the simulation is given as:

$$P(S) = 30 \cdot d(S) + \frac{1}{T(S)} \sum_{t<T(S)} p(t) \tag{4.2}$$

A simulation in which all units die terminates prematurely and sets 1000 as its performance.

Solution strategy

To prevent unit collisions, we need units to repulse each other. In order to position them on the line, they need to be attracted to it. This gives us the logical division that all the units will be in the same group and the line will have its own group. We will set one force law that works internally among units and we will set a force law that acts as an attractor between the units and the line. The line is not affected by any potential fields, it is scripted to move in a certain way. It acts as a virtual leader.

The pull from units to the line should scale with their distance to it, such that units that are far away are pulled harder than units that are close. To ensure

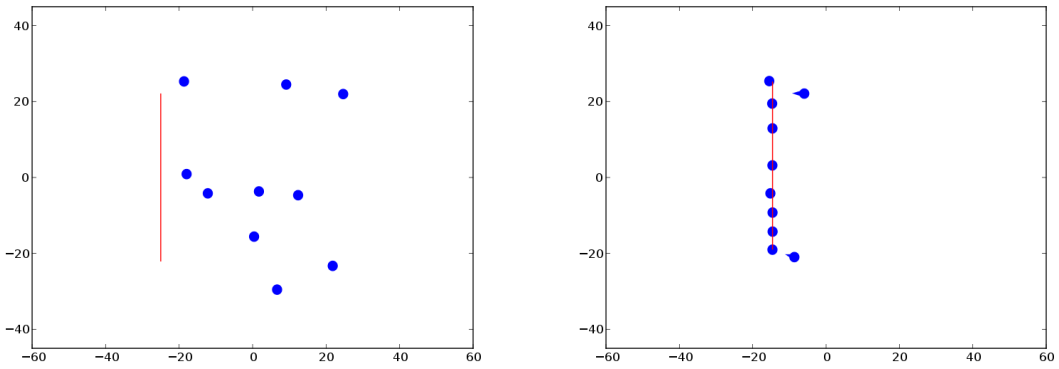

Figure 4.1: The initial distribution of units for the line formation experiment and a distribution of units at a later time in the simulation.

that units that get close to one another are repulsed, we will scale the magnitude of forces between them inversely with their distance. Let $r(i,j)$ be the distance between units i and j, σ_1 and c_1 be positive constants. Then the magnitude of the force between the units will be given as:

$$f(i,j) = \frac{c_1}{r^{\sigma_1}(i,j)} \tag{4.3}$$

This is a social potential force law. Because c_1 is positive, the force is a repulsive one. Because σ_1 is also positive, the force increases as units get closer. To add direction to this force, we simply multiply it by the unit vector $\vec{u}(i,j)$ that is the direction from unit i to unit j.

Let $r_l(i)$ be the distance from unit i to the line and c_2 be a negative constant and σ_2 a negative constant. Using a social potential force law again, we get:

$$a(i) = \frac{c_2}{r_l^{\sigma_2}(i)} \tag{4.4}$$

Because $c_2 < 0$, this is the magnitude of an attraction between unit i and the line. Since we also require that $\sigma_2 < 0$, this magnitude scales positively with the distance r_l. To add direction to this force, we multiply it by the unit vector $\vec{u}_l i$ which is the direction from unit i to the nearest point on the line. In total, the force on a unit i is then given as:

$$\mathbf{F}(i) = \vec{u}_l(i)a(i) + \sum_{j \neq i} f(i,j)\vec{u}(i,j) \tag{4.5}$$

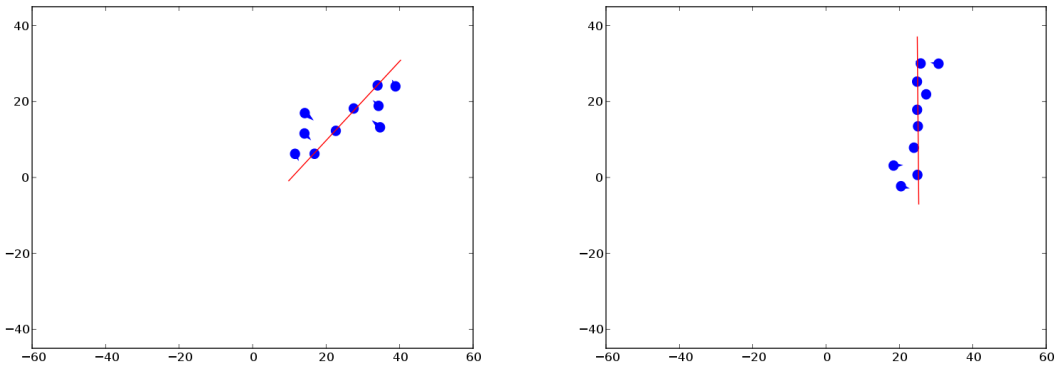

Figure 4.2: First revolution and final configuration of line experiment with handcrafted parameters.

Result with handcrafted potential functions

Reiterating the meaning of parameters, c_1 is the coefficient that constitutes the magnitude of the repulsive force between units, it is divided by their distance raised to σ_1. c_2 is the coefficient that constitutes the magnitude of the attraction between units and the line and it is divided by that distance raised to σ_2. Because there are 10 units and the line is 44 long, setting $c_1 = 6$ seems a good place to start. Because we do not want the repulsive force to dominate too much, we set $\sigma_1 = 2$. We want the units to be attracted to the line, but we do not know how strongly yet, so we set $c_2 = -1, \sigma_2 = -1$.

We observe that two units die in this case, so we require them to push each other earlier or more strongly. Clearly, there exists a case where the pull from the line dominates the repulsion between units. Setting $\sigma_1 = 1$ fixes this particular case. This is already a good result - no units die at this point. It seems unlikely that this is optimal, though. Table 4.1 is presentation of a sequence of attempts to better the performance. Figure 4.2 is a visualization of the last run. Finding potential functions that solve this problem is clearly not too difficult.

c_2	σ_2	c_1	σ_1	Survived	Performance
-1	-1	6	2	8	63.254
-1	-1	6	1	10	5.27562
-1	-1	8	2	8	63.3336
-1	-1	8	1.5	10	4.28025
-0.5	-1	6	2	10	3.60628
-0.5	-1	5	2	8	63.4308
-0.5	-0.5	7	2.3	10	3.51985

Table 4.1: Some experimental results with handcrafted potential functions

Using optimization techniques

From Table 4.1 it seems very clear that there exist many good solution in the region bounded by:

$$0 < c_1 \leq 14{,}0 < \sigma_1 \leq 3$$
$$-2 \leq c_2 < 0, -2 \leq \sigma_2 < 0$$

Recalling from Subsection 3.3, this means that the majority of the initial population will be generated inside that range. The genotype is simply a 4-element vector of reals that correspond to these parameters. With one exception, all default values explained in Subsection 3.3 will be used. We make the exception that a gaussian_stddev of 1 is too high, since the weights we use are so small and so adjust it to 0.3 to prevent very large mutations from happening. Furthermore, because the genome is very short, we raise the mutation rate for the simulated annealing trial from 0.1 to 0.25. For all of the optimization algorithms, regardless of population size, we limit the amount of fitness evaluations allowed to 240.

To ensure that we will not simply get lucky with the initial population, each optimization technique will be run 10 times with a different initial population each time. The evolutionary strategy and genetic algorithm will both be run with a population size of 40, whereas particle swarm optimization will be run with a population size of 12 and simulated annealing will run with only one candidate in its population. We can use these low population sizes because we have bounded the region and generate the initial population in a way that makes it very likely that at least some individuals in the initial population will have a good fitness.

Results of optimization techniques

Table 4.2 shows different statistics for the best individual generated by the 10 optimization runs for each optimization technique that was used. We note that PSO seems exceptionally well-suited for this particular optimization problem - it has the strongest median, average and best case. ES has by far the lowest variance in results and consistently produce very good potential functions. SA is notable for producing the highest variance - undoubtedly, the other optimization techniques utilize their ability to explore many areas in the search space in parallell well. While these performances are not exceptionally much stronger than the ones generated in much less time by trial and error, it is worth noting that all runs produced potential functions which ensure the survival of all units.

Table 4.3 shows the best individual produced by the best run, median run and worst run for each optimization technique. It is interesting to see the variety in

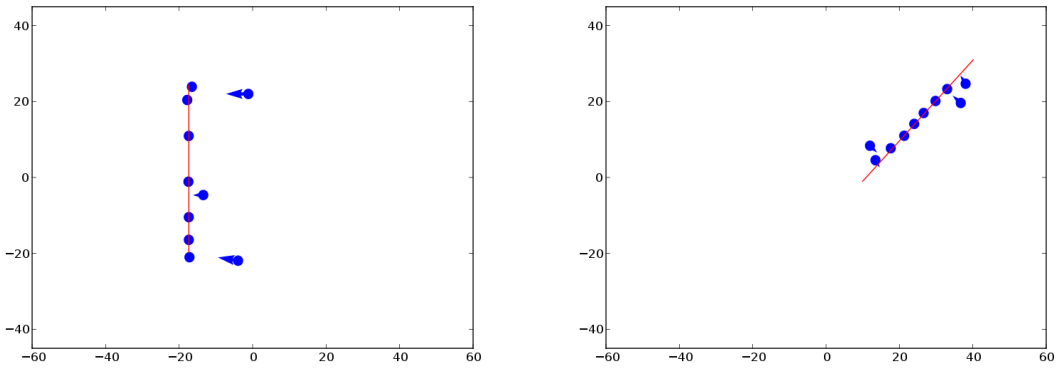

Figure 4.3: Snapshots of line experiment with machine-learned weights for potential fields. Created while adapting simulations to inspyred. These weights were found by an ES.

Technique	Best	Average	Median	Worst	Variance
ES	3.23154	3.35999	3.368	3.56088	0.00832085
SA	3.26314	3.56375	3.51894	4.61347	0.142634
PSO	3.0933	3.25616	3.19977	3.79004	0.0385233
GA	3.32057	3.60986	3.57893	4.16558	0.0550932

Table 4.2: Statistics of best individuals found by different optimization techniques through 10 runs.

weights that are applied. All these individuals create potential fields for which all units survive, despite how different some of them are. This reinforces the importance of the concept of equilibrium distance - it is the relationships between the weights that matter, not their absolute magnitudes.

Conclusion

We have shown that we can make units that work under conditions similar to those found in games march in a line formation using artificial potential fields. Furthermore it is possible to make them spread relatively evenly along their line formation. The units in this experiment adapt relatively well when the line rotates faster than they can move in most cases.

Optimization techniques can discover good potential functions for this type of formation. Although they do not in all cases result in potential functions that perform *better* than parameters discovered by trial and error they all result in potential functions that achieve the objective of the experiment. This is promising because it may be harder to discover potential functions for experiments that demand more parameters.

Technique	c_2	σ_2	c_1	σ_1	Performance
ES best	-0.666186	-0.480902	9.60076	2.62909	3.23154
ES median	-0.564949	-0.96016	8.29156	2.40485	3.368
ES worst	-0.206973	-0.1	5.6278	3	3.56088
SA best	-0.409996	-0.333396	10.2848	3	3.26314
SA median	-0.4752	-2	6.85887	2.05795	3.51894
SA worst	-2	-1.11039	8.39082	1.04318	4.61347
PSO best	-0.992032	-0.334872	14	2.87928	3.0933
PSO median	-0.907889	-1.20949	14	2.62581	3.19977
PSO worst	-0.324438	-2	5.61709	1.85067	3.79004
GA best	-0.8736	-0.960802	10.4985	2.33163	3.32057
GA median	-0.92958	-1.37381	9.74959	1.88497	3.57893
GA worst	-1.36468	-1.39426	9.72775	1.39983	4.16558

Table 4.3: Best genomes found by 10 runs of 4 optimization techniques.

All optimization techniques that were tested for this experiment produce good, usable potential functions with a high degree of reliability. Evolutionary strategies stand out in this experiment by producing a worst case that is comparable to the median cases found by genetic algorithms and simulated annealing. Particle swarm optimization stands out in its best few cases, recording the highest performing set of parameters and the highest performing median.

Creating a box formation

Objective

Create a simulation of units that gather together in a compact box formation. The units are initially randomly distributed in a small area and need to converge to a box formation quickly.

Environment

The environment does not have any obstacles. Units have a simulated friction to limit their speed. Their velocity is calculated by adding acceleration to it at every timestep. Their acceleration is calculated from potential fields. Units are not allowed to collide.

Motivation

Strategy games commonly require players to pack their units into small areas. This is also a very space-efficient formation to use for troop movements and it is a good default formation from which it is possible to go into other formations quickly.

Performance Measure

The average area of the smallest square that can fully contain all units for each timestep divided by the amount of space occupied by units.

Expected Outcome

A setup of potential functions that guides units into an approximately square formation quickly. Optimized potential functions are expected to pack more tightly and more quickly than handwritten ones.

Description

Like in the line experiment, this experiment takes place in a 2-dimensional world with a contineous coordinate system. Also similar is that the world is not bounded, but the initial distribution of units is inside a 60 by 60 box. For this experiment, 20 units will be placed inside this box using a random uniform distribution with the constraint that no unit can be placed within a distance of 5 from another. Units are circular with a radius of 2, such that they crash when the distance between them is ≤ 4. Units that collide will be removed from the simulation. Instead of employing a speed limit for units, we will use a hook that slows down their speed to $\frac{3}{4}$ of what it was, at each tick. This simulates a friction effect and means that it will take a large force for units to move very quickly.

Because each unit requires an area of $\pi r^2 = 4\pi$ at least, we know that the formation can pack no tighter than $80\pi \approx 251.33$ unless units die. The smallest square in which 20 circles of radius 4 can be packed, has an area of 322.423922761 ([Nurmela and Östergård, 1997]), corresponding to a side length of 17.9561667056.

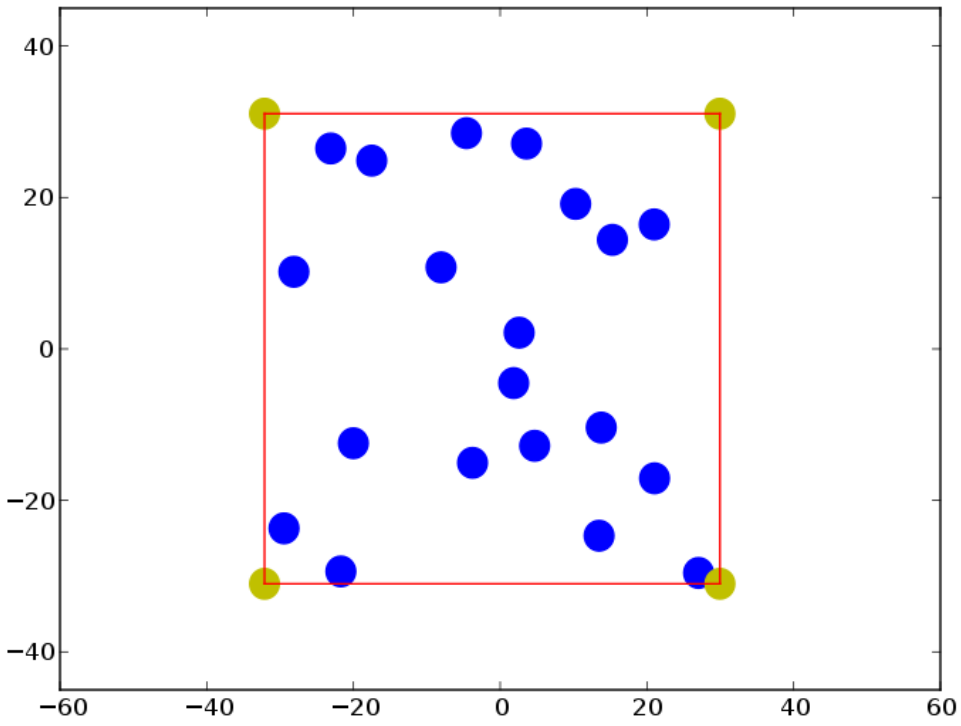

Figure 4.4: Initial distribution of units and the area of their bounding square in the box experiment.

The simulation will run for 3000 timesteps, each lasting 0.2 simulated seconds.

Performance Measure

Let t denote the current timestep, $A(t)$ the area of the smallest square that fully contains all living units at time t. Let $d(t)$ be the amount of dead units at time t and $n(t)$ be the amount of living units. We choose 15^2 as the penalty for a unit dying, because it corresponds to a relatively large expansion of the bounding square. We divide this by the known minimum area occupied by this many units to get out a number that relates to how far we are from the optimal packing. We let $p(t)$ be the performance of the simulation at time t:

$$p(t) = \frac{A(t) + 225d(t)}{n(t)4\pi} \tag{4.6}$$

Let $T(S)$ be the amount of timesteps t in the simulation instance S. Then the

performance of S is given as:

$$P(S) = \frac{1}{T(S)} \sum_{t<T(S)} p(t) \tag{4.7}$$

This performance measure is designed to be minimized. We choose to pack units into a box instead of a different shape (like a disc) because most real-time strategy games use *isometric* worlds.[1]

Solution strategy

There are three different types of behaviour we need to encourage. Units can not collide, so they need to repulse one another. They need to attract one another, or they all need to be attracted to the same area. Finally, the shape of their formation should efficiently utilize the area of a square.

Units governed by simple attractive-repulsive force law pairs, have as we have seen in Figure 2.7 and [Reif and Wang, 1999] a tendency to form clusters shaped like *discs*. A disc is not really what we are aiming for, although it is a space-efficient packing of the units. To make a disc formation into a box formation, what is needed is to add attraction to the areas where the corners of the square would be. This presents a problem, however. Supposing that the pull to the corners scale inversely with the distance to it. Then, units near the center of the box could be almost unaffected by it. However, units that come very close receive extreme levels of attraction. Supposing instead that we let the attraction scale with the distance. Then units that are far away get pulled towards the corners very strongly as well. Furthermore, units will seek to the areas in which the attraction from the corners cancels out. This only happens in the center of the formation and we are back to where we started.

Attracting units to the corner as a function of how far away they are is still a promising idea, however. This is because units that are close to the corners are affected only very weakly, which means that they can safely go all the way into the corner and fill up the square. Recalling 2.2, it is not unusual to design potential functions that have a minimum distance of influence, ρ_0. We can use this to ensure that only units that are already within the relevant area of the square are pulled towards corners. We are already calculating the bounding square of the formation and it is trivial to add virtual leaders to its corners to pull units in that direction. We will let ρ_0, the minimum distance of influence for the virtual leaders, be dependent on the side length $s(t)$ of the bounding square at time t such that $\rho_0(t) = \frac{s(t)}{2}$. In other words, each corner of the bounding square acts like a virtual

[1] Worlds that consist of tiles of some shape, typically square. In other words, the terrain obstacles are usually built from rectangles.

leader that attracts units in a circle with radius equal to half the side length of the bounding square.

Going back to how clustering force laws work, we require that attraction dominates repulsion when units are far away from one another. Let r be the distance between the units and $\vec{u}$ be the unit vector that gives the direction between two units, then the force law $f(i,j)$ between units i and j is given as:

$$f(i,j) = \vec{u}\left(\frac{c_1}{r^{\sigma_1}} + \frac{c_2}{r^{\sigma_2}}\right) \tag{4.8}$$

By choosing c_1 to be negative, that term is the attractive part of the force law and we require that $\sigma_1 < \sigma_2$ to make it dominate for when r is large.

We will define a similar force law between the virtual leaders and the units that should form a box formation. Let ρ_0 be the minimum distance of influence of a virtual leader, $\vec{u}$ be the unit vector that gives the direction between a unit and the virtual leader, and r be the distance between them. Then the force between the virtual leader j and the unit i is:

$$f(i,j) = \begin{cases} \vec{u}c_3 r^{-\sigma_3} & \text{if } r < \rho_0 \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

By setting c_3 and σ_3 negative, units that are far away from the corners will be attracted to them more than units that are close. The force on each unit in the group is the sum of forces from the other units in its group plus the sum of forces from the virtual leaders.

Result with handcrafted potential functions

The adjustment of parameters and the results they gave are documented in Table 4.4. We have decided on which of the parameters for the experiment to set negative and which to set positive. At this point, we need to decide on the magnitudes and try to discover parameters that not only produce a box with a small area but also let as many units as possible survive. We reason that it is better to start on the safe end, so we decide to set $c_1 = -1, \sigma_1 = 1$ and $c_2 = 12, \sigma_2 = 2$ as a starting point.

For the time being, we do not let the virtual leaders at the edges of the formation exert any forces. Keeping the values for the attractive part of the clustering force law constant, we reduce c_2 until we get our first collision. At this point, we attempt to make the virtual leaders in the corners of the bounding square come into play. Because we decided to set σ_3 negative, we need to be careful with the

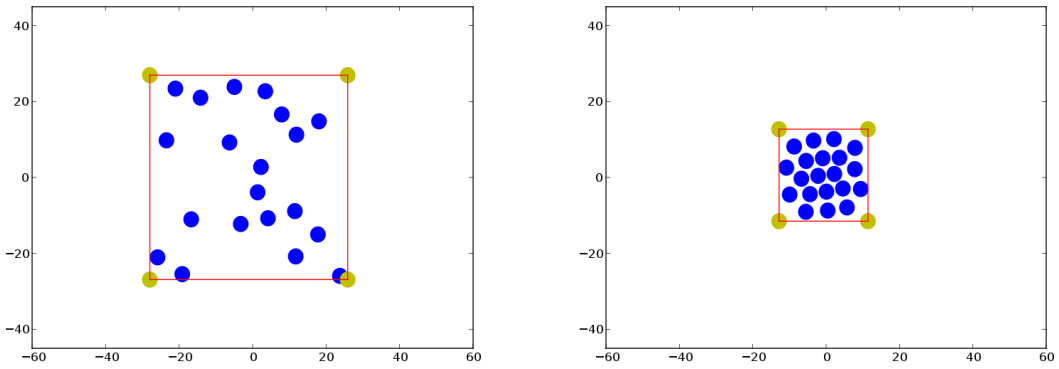

Figure 4.5: Early and final configuration of the box experiment with the best parameters from Table 4.4.

c_1	σ_1	c_2	σ_2	c_3	σ_3	Survivors	$P(S)$
-1	1	12	2	0	-0	20	5.0227
-1	1	10	2	0	0	20	3.96564
-1	1	9	2	0	0	18	4.66045
-1	1	9	2	-0.05	-0.05	20	3.69161
-1	1	9	2	-0.05	-0.025	20	3.68374
-1	1	9	2	-0.025	-0.025	16	6.87506
-1	1	9	2	-0.0375	-0.05	20	3.63013

Table 4.4: Experimental results with handcrafted potential functions for the box formation experiment.

magnitude of both that and c_3, as they scale positively with the distance from the corner to the unit - meaning that they can quickly dominate the clustering force law and stretch the formation. The area of the bounding square for the final frame of the last simulation run has an area that is 588 - only 1.82 times larger than the optimal packing of circles into a square.

Using optimization techniques

The genome for this optimization problem is a vector of 6 reals:

$$[c_1, \sigma_1, c_2, \sigma_2, c_3, \sigma_3]$$

Because of the complex interactions between these parameters and the fact that there are 6 of them, it is much more difficult to bound the search space for this problem than for the line experiment. We make the assumption that large absolute values for the forces from the virtual leaders will stretch the formation, so we set the bounds for those to $-0.8 < c_3 < 0, -0.8 < \sigma_3 < 0$. We also require that

$c_1 < 0$ to create attraction between the units and that $c_2 > 0$ to create repulsion. We set $-2.5 < c_1 < 0$ and $0 < c_2 < 20$. We do not know whether we want the magnitude of the internal forces to scale positively with distance or negatively, so we will allow both by setting $-2 < \sigma_1 < 2$, $-1 < \sigma_2 < 4$. To summarize:

$$\begin{aligned} -2.5 < c_1 < 0, -2 < \sigma_1 < 2 \\ 0 < c_2 < 20, -1 < \sigma_2 < 4 \\ -0.8 < c_3 < 0, -0.8 < \sigma_3 < 0 \end{aligned}$$

For this experiment we will use the default parameter values provided by inspyred again, with the modification that we set gaussian_stdev to 0.1 to avoid large mutations. For the simulated annealing runs, population size will again be 1, while it will be 20 for evolutionary strategies, 12 for particle swarm optimization and 40 for genetic algorithms. Each optimization technique will be allowed to evaluate the performance of 240 simulations. Again, we make the exception that for simulated annealing we set mutation_rate differently, this time to 0.2. Each optimization will be run 10 times, with different initial populations.

Results of optimization techniques

Tables 4.5 through 4.8 document best, median and worst optimization runs for this experiment. Table 4.9 documents statistics for the 10 runs of each optimization technique.

c_1	σ_1	c_2	σ_2	c_3	σ_3	$P(S)$
-1.12931	0.173039	17.3137	1.31857	-0.532291	-0.378826	2.24204
-1.06658	0.731205	10.3184	1.72049	-0.0691291	-0.485037	2.95752
-0.0289707	-0.649268	9.72028	1.36011	-0.172005	-0.332695	4.87545

Table 4.5: Best, median and worst optimization result by PSO on the box experiment.

The results from applying PSO to the problem are impressive, again. However the impressive performances are mostly due to quickly packing into a box, not so much because the units pack into a very small box. The area of the bounding square for the best PSO run is $23.18^2 = 537.31$, 1.66 times as big as optimal packing of the the units in a square. What's more impressive is the pace at which this square is formed. As we can see in Figure 4.6, we get a very good approximation of a box formation already at timestep $t = 46$ of the simulation and it is almost static from timestep $t = 412$. It takes more than twice as long for the best result from Table 4.4 to reach its stable configuration.

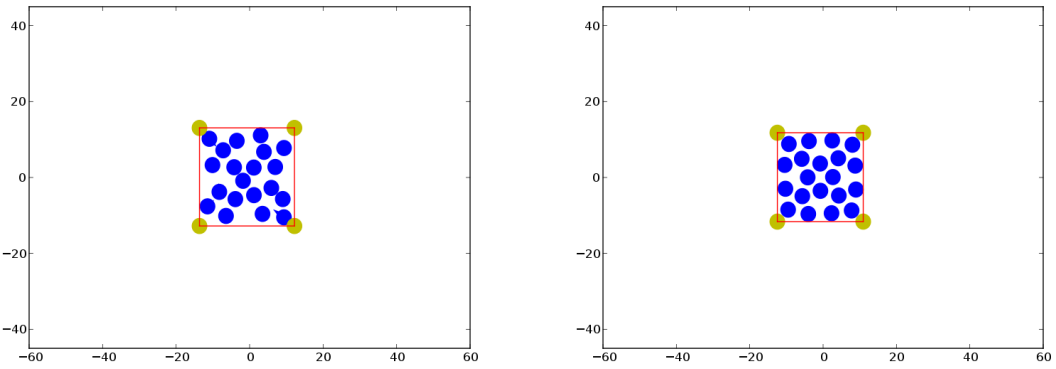

Figure 4.6: PSO best result at timesteps 46 and 412.

c_1	σ_1	c_2	σ_2	c_3	σ_3	$P(S)$
-0.936459	0.285367	14.123	1.4454	-0.37234	-0.294172	2.23663
-0.982124	0.179539	8.51833	1.0619	-0.322346	-0.543172	2.46691
-0.723981	0.0387806	9.51887	0.912572	-0.421388	-0.468015	5.41593

Table 4.6: Best, median and worst optimization result by GA on the box experiment.

On this experiment, the genetic algorithm performed much better than all the other optimization techniques. This is a little surprising in context of the line experiment, where it had the worst best result, the worst average and the worst median. Even more surprising is the fact that it seems to do this despite often throwing away good results in early generations (See Figure 4.7). An obvious enhancement to the results found by the GA would be to keep a store of the best individual found so far or to use elitism to ensure that it would not perish. The figure also suggests that running more iterations could improve the results.

c_1	σ_1	c_2	σ_2	c_3	σ_3	$P(S)$
-0.001	-1.38582	9.00403	2.34396	-0.001	-0.001	2.66467
-1.28216	0.0973462	8.14626	0.761273	-0.260933	-0.296327	3.94356
-0.001	-0.754197	11.792	2.82204	-0.001	-0.001	5.47226

Table 4.7: Best, median and worst optimization result by ES on the box experiment.

Simulated annealing again suffers from its inability to search the search space in parallell. It seems much more dependent on having a good starting position for the search than the other optimization techniques do. This results in it finding very good results once in a while, but its average result is by far the worst. It is possible that this is because the cooling schedule used is not very good. However, the other optimization techniques required almost no tweaking to produce strong results. Because the runtimes are almost identical (recall that they are all allowed

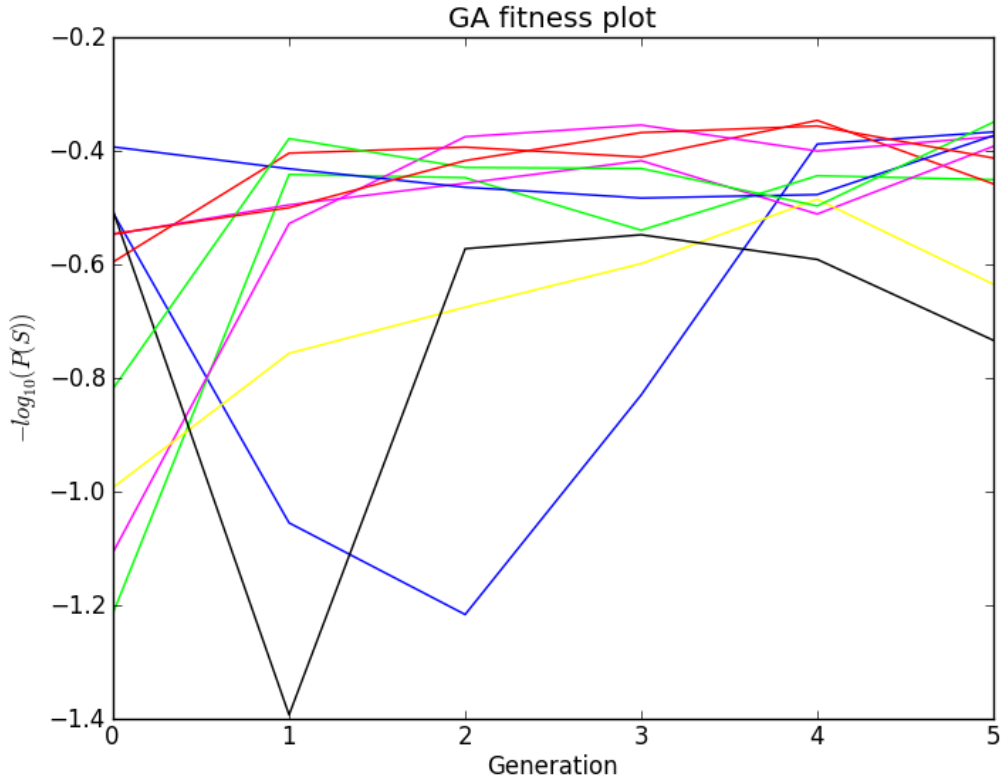

Figure 4.7: The progression of best individual fitness for each GA run.

the same amount of simulation evaluations), this speaks in favor of not using simulated annealing for this particular problem.

c_1	σ_1	c_2	σ_2	c_3	σ_3	$P(S)$
-0.069546	-0.369282	11.7752	1.83189	-0.377307	-0.381649	2.22552
-0.931956	1.48086	14.5459	2.82628	-0.001	-0.342177	7.26702
-1.41064	0.311503	15.2475	3.07503	-0.424079	-0.429368	251.39

Table 4.8: Best, median and worst optimization result by SA on the box experiment.

With the exception of the 6 worst of the simulated annealing results, all potential functions developed here result in a very good approximation to a square formation. It is also interesting to note the internal structure in the formations. The most common internal shape seems to be one for which it is easy to transition into a wedge formation. The centermost units typically form a diamond shape. Extending this shape to the left, right, top or bottom of the shape results in a triangle with the side as its base. Figure 4.6 illustrates this type of internal shape. This has a structural resemblance to the optimal packing of equal circles in a square.

Technique	Best	Average	Median	Worst	Variance
PSO	2.24204	3.05471	2.95752	4.87545	0.476711
GA	2.23663	2.97767	2.46691	5.41593	0.989851
ES	2.66467	4.00748	3.94356	5.47226	0.663325
SA	2.22552	37.3505	7.26702	251.39	5202.29

Table 4.9: Statistics for best individuals found through 10 runs of each optimization technique on the box experiment.

Conclusion

We can conclude that artificial potential fields can be used to put units in strategy game like environments into box formations. The box formations produced do not resemble typical grid-like box formations, but their outer shape is close to square. Units can be packed tightly and quickly to maintain a square shape that is a good default formation from which to spread into looser formations, line formations or wedges.

Optimization techniques can be applied to potential field functions to produce novel formations. All four optimization techniques applied to the problem can produce good results with very reasonable reliability. Because of the performance measurement used for this experiment, most parameters that result in good performance are focused around moving the units into a square formation *fast*. The weights discovered are thus not necessarily the best for units *tightly* or safely.

Genetic algorithms with blend crossover and particle swarm optimization produce very good potential functions for this formation with a very high degree of reliability.

Creating a wedge formation

Objective

Create a simulation of units that gather together in a wedge formation. The units are initially distributed in a box formation.

Environment

The environment does not have any obstacles. Units have a simulated friction to limit their speed. Their velocity is calculated by adding acceleration to it at every timestep. Their acceleration is calculated from potential fields. Units are not allowed to collide.

Motivation

Wedge formations are useful for breaking and splitting enemy formations to isolate enemy groups of units from one another.

Performance Measure

The fraction of units inside the predetermined wedge shape averaged over each timestep.

Expected Outcome

A setup of potential functions that guides units into an approximately wedge formation, similar to a filled v-shape. Optimized potential functions are expected to pack quickly.

Description

The experiment takes place in the same world and the same conditions as in the box experiment (see 4.1). To be more precise, the final frame of the box experiment instance that was found by the best PSO run is used as the first frame of this experiment. The rules of this simulation are identical, units are slowed down to $\frac{3}{4}$ of their speed at every simulation timestep, they are circular with a radius of 2 and they are removed from the simulation if they crash. The initial distribution of units can be seen in 4.1. The simulation will run for 3000 timesteps, using a timedelta of 0.2.

A wedge formation is vaguely like a triangle. It is typically employed to break through enemy lines. The tip of the wedge is often heavily reinforced. Because of its v-shaped front, it is hard to do flanking manouvers on a wedge - the distance to get behind it is long. To give the wedge punching power, the main weight of units need to be positioned behind its tip.

Because of its similarity to a wedge, we use a triangle to delimit the area in which we'd like the units to be. The height of this triangle is equal to the side length of the bounding square for the units at the start of the simulation. The

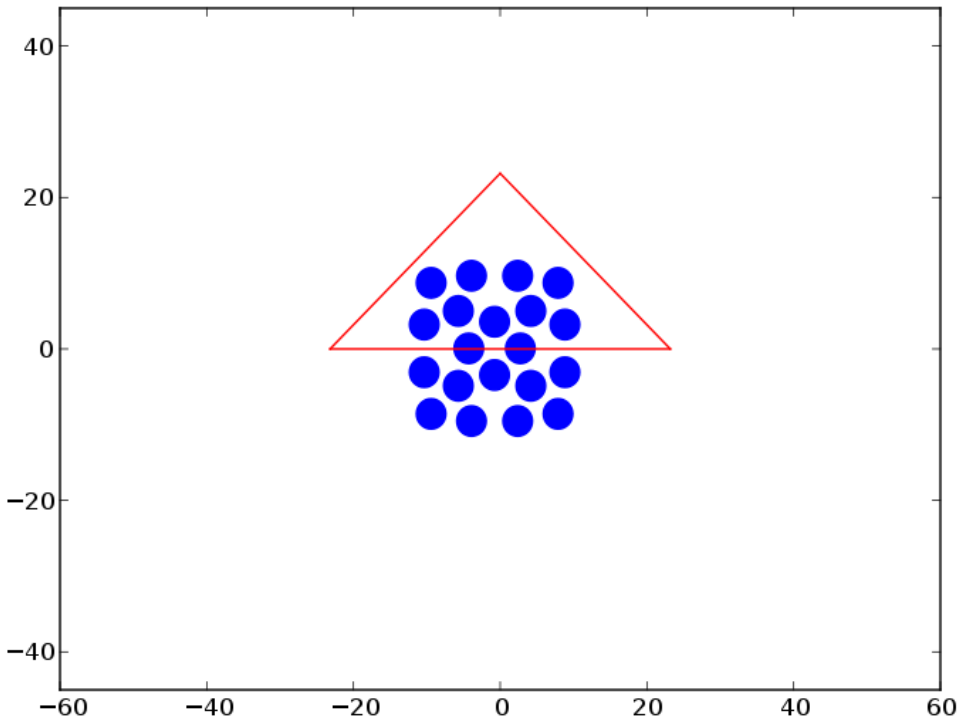

Figure 4.8: Initial distribution of units for the wedge experiment. The red lines show the shape of the wedge formation.

length of the base of the triangle is 2 times as long. Primarily, we are looking to fill the center of this triangle with units in a spearhead-like formation. The center of the base of the triangle is located in $(0, 0)$.

Performance Measure

Let t be the current timestep in a simulation. Let $a(t)$ be the amount of units inside the wedge shape at time t and n be the amount of units that were added to the simulation. The performance of the simulation at timestep t is $p(t)$ as in (4.2).

$$p(t) = \frac{a(t)}{n} \tag{4.10}$$

Let $T(S)$ be the amount of timesteps in simulation S. Then the performance of

the simulation is:

$$P(S) = \frac{1}{T(S)} \sum_{t<T(S)} p(t) \tag{4.11}$$

This performance measure is designed to be *maximized*. It is the average fraction of units that are inside the wedge shape at any timestep.

Solution Strategy

We require the following of our units to create a good wedge formation:

- The units should not collide.
- The largest density of units should be below the tip of the wedge.
- The frontline of the units should be approximately shaped like an upside-down v.

To avoid collisions, we will use a repulsive force law between units. For the other two behaviours, we will attempt to use attractive virtual leaders. It is not immediately obvious how to use virtual leaders to accomblish this. The simple approach of adding virtual leaders to the corner of the triangle can not give us the desired v-shape we are looking for. This is because the minimum with this layout does not look like a wedge at all, see Figure 4.9. This plot is taken with an attractive force law with $c = -1, \sigma = -0.25$. Using attractors as edges of the triangle gives a better shape. Even so, we can tell that while the minimas are on the lines of the triangle, these spots will be filled by some units and then their potential will rise. When that happens, there will be minimas both outside and inside the triangle.

What we need to do is to make the minimas be located inside the triangle, not at its edges. Using virtual leaders that are lines from the corners of the triangle to its center of mass gives us the shape in Figure 4.10. This is a good shape to base our formation on. The center of mass of the triangle is the most attractive point in the shape, which should make the formation dense behind the tip of the wedge. Additionally, once the local minima present along the lines are filled, the new minimas will still be closer to the interior of the triangle than in either of the other solution strategies. These lines are using round edges.

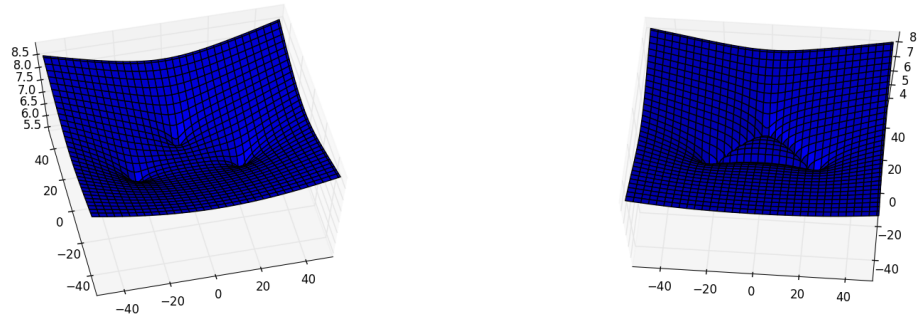

Figure 4.9: On the left, potential field with 3 attractive virtual leaders in the corners of a triangle. On the right, using virtual leaders as the edges of the triangle.

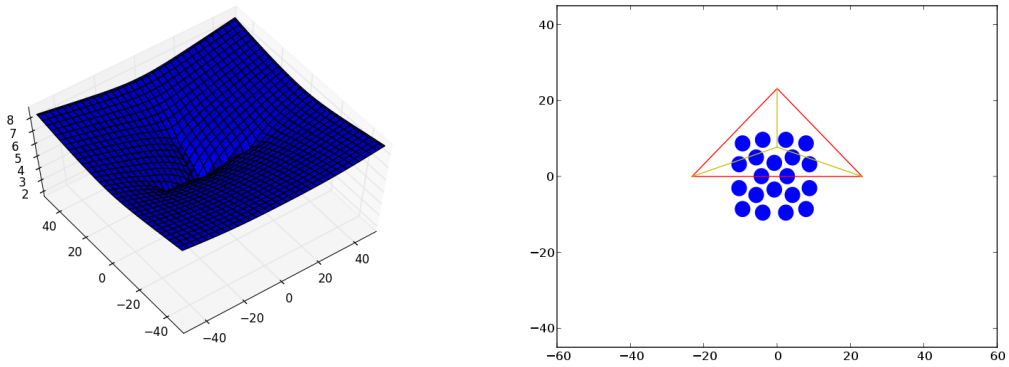

Figure 4.10: Using edges from the corners of the triangle to its center of mass as attractors.

We let $r(i,j)$ be the shortest distance from i to j and $\vec{u}(i,j)$ be the direction between them. Let v be a virtual leader and j a unit. Then the force on each unit i is:

$$\mathbf{F}(i) = \sum_{j \neq i} \left(\vec{u}(i,j) \frac{c_1}{r^{\sigma_1}(i,j)} \right) + \sum_{v} \vec{u}(i,v) \left(\frac{c_2}{r^{\sigma_2}(i,v)} \right) \tag{4.12}$$

We require that $c_1 > 0$ to create repulsion between units and we require that $c_2 < 0$ to create attraction to the virtual leaders. We also require that $\sigma_1 > 0$ so that repulsion is stronger when units are close than when they are far away. We need $\sigma_2 < 0$ to make attraction stronger when they are far away than when they are close.

c_1	σ_1	c_2	σ_2	Survivors	$P(S)$
10	1	-0.5	-0.5	20	0.355525
10	1	-0.5	-0.75	20	0.515475
10	1	-0.75	-0.75	20	0.700125
10	1	-0.9	-0.75	0	0
9	1	-0.75	-0.75	14	0.699375
9	1	-0.75	-0.7	20	0.710225
9	1	-0.8	-0.7	20	0.790625
9	1	-0.25	-1	20	0.4785
9	1	-0.5	-1	20	0.977375
9	1	-0.525	-1	14	0.70035

Table 4.10: Handcrafted potential function parameters for the wedge experiment.

Results with handcrafted potential functions

Table 4.10 documents a series of attempts to better the performance of the wedge experiment. For the first attempt, we set parameters such that we are reasonably sure that the repulsive force between units will dominate the attraction to the virtual leaders. This gives us a formation that is much larger than the triangle the units should try to occupy, as expected. Increasing the attraction to the virtual leaders reduces this in size. Once we have set $c_2 = -0.9, \sigma_2 = -0.75$, the attraction dominates too much and all units die. We attempt to go back to an earlier set of parameters and this time reduce the repulsion very slightly instead of increasing attraction. This is also enough to cause some units to die, although the surviving units all stay inside the triangle.

Trying instead to reduce the coefficient of the attractive force signicantly and instead let it scale linearly with distance seems like a promising path. Bumping up the coefficient for the attraction leaves us with a very good result. Trying to increase it much further causes units to die again. The best result from the handcrafted potential functions leaves the final configuration documented in Figure 4.11.

This is a very good wedge formation. The core of the formation is well protected by a v-shaped front and the tip of the formation has the large weight of units that is called for.

Using optimization techniques

The genome for this optimization problem is a vector of 4 reals:

$$[c_1, \sigma_1, c_2, \sigma_2]$$

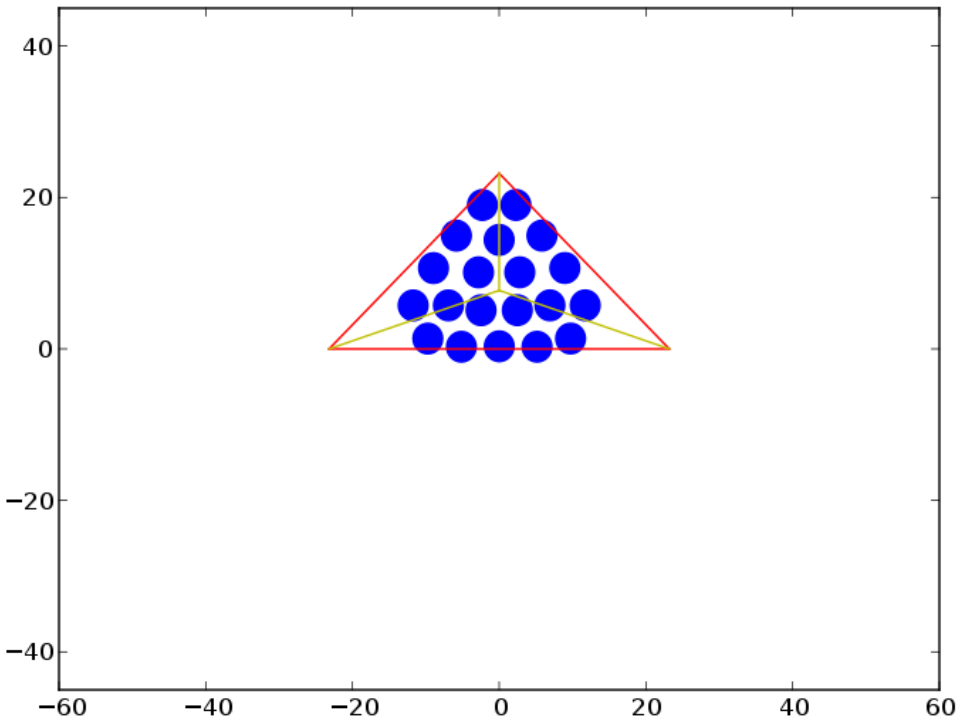

Figure 4.11: The final configuration of the wedge experiment with the best parameters that were found by trial and error.

Because we have already found an extremely strong solution, we could narrow the search space down to the bounds around that. However it is more interesting to see if perhaps there exist more areas with good solutions and furthermore whether the optimization techniques will be able to locate it without much assistance. Therefore, we will bounds that are a little spread:

$$0 < c_1 < 20, 0 < \sigma_1 < 2$$
$$-2 < c_2 < 0, -2 < \sigma_2 < 0$$

We will use the parameter values provided by inspyred, excepting the value for gaussian_stdev, which we set to 0.1. For simulated annealing, we will set mutation_rate to 0.25. The population sizes are 1 for simulated annealing, 12 for particle swarm optimization, 40 for evolutionary strategies and 40 for genetic algorithms. We will allow 300 fitness evaluations for each technique for each run. We will perform 10 runs with different initial populations for each technique.

Technique	Best	Average	Median	Worst	Variance
SA	0.966325	0.649028	0.722225	0.20285	0.0550979
GA	0.988775	0.961755	0.973575	0.88975	0.000975161
PSO	0.9774	0.797828	0.861475	0.41975	0.0394358
ES	0.886	0.79281	0.8509	0.6067	0.0101037

Table 4.11: Optimization result statistics for the wedge experiment.

c_1	σ_1	c_2	σ_2	$P(S)$
4.57825	1.22634	−0.183973	−0.929281	0.966325
10.0242	0.364445	−1.64309	−1.04323	0.722225
5.54302	0.0139136	−0.49841	−1.74024	0.20285

Table 4.12: Simulated annealing optimization results for the wedge experiment.

c_1	σ_1	c_2	σ_2	$P(S)$
12.7609	1.08366	−0.696471	−0.914919	0.988775
11.6994	0.899228	−0.803483	−0.976608	0.973575
10.6517	0.815878	−0.9832	−0.892773	0.88975

Table 4.13: Genetic algorithm optimization results for the wedge experiment.

Results of optimization techniques

Tables 4.12 through 4.15 document the best, median and worst optimization runs for this experiment. Table 4.11 documents statistics for the optimization runs.

All optimization techniques are able to find good potential functions for this problem. None of the results do much better than the handcrafted functions, which is not unexpected.

Simulated annealing has the worst average performance, the highest variance and also a bad worst case. This stands in stark contrast to the genetic algorithm, which performs exceptionally well on this experiment. Its average case is only marginally worse than the parameters found by hand were and its worst case is better than the best case of evolutionary strategies on this problem.

Particle swarm optimization has the second best average case and it achieves a very good best case as well. However, its worst case is worse than that of both evolutionary strategies and genetic algorithms.

On the whole, most of the results obtained by these optimization runs give satisfactory wedge *shapes*. They are not necessarily contained inside the triangle provided to the simulation, however and in some cases units collide.

c_1	σ_1	c_2	σ_2	$P(S)$
12.8845	0.856295	−0.892512	−1.02084	0.9774
12.0033	0.681236	−0.855669	−1.15603	0.861475
6.23697	1.08907	−0.482609	−0.38614	0.41975

Table 4.14: Particle swarm optimization results for the wedge experiment.

c_1	σ_1	c_2	σ_2	$P(S)$
14.3339	1.21731	−0.520555	−0.907055	0.886
13.6439	0.705424	−0.99712	−1.1048	0.8509
6.83697	2	−0.168134	0	0.6067

Table 4.15: Evolutionary strategies optimization results for the wedge experiment.

Conclusion

We can conclude that potential fields can be used to position units in a strategy game like environment into wedge formations. The wedge formations produced adhere closely to the desired outcome of a grouping of units with a v-like frontline and a thick concentration of units behind the tip of the shape.

Optimization techniques can produce good potential functions to create this type of formation. Genetic algorithms with blend crossover produce exceptionally good results very reliably.

4.2 Robustness testing

Robustness of line formation

Objective

Discover whether experimental results from previous experiments can be applied to more complex environments.

Environment

The environment has patches of terrain where units move more slowly. Units can move at a limited speed that differs from unit to unit. Their velocity is calculated by adding acceleration to it at every timestep. Their acceleration is calculated from potential functions found in 4.1.

Motivation

The utility of potential fields would be much greater if it is possible to use simple environments to create formations that work in a vide variety of environments.

Performance Measure

Minimize the amount of units that collide and the average distance from units to the line that is the center of their formation.

Expected Outcome

Tight clumps of units and possibly collisions near the back end of the line when it moves through difficult terrain. The line shape is also expected to be ragged while moving through terrain where only some units will be slowed.

Description

This experiment takes place in a world that is almost exactly like the world in the line experiment. The difference is that there are now patches of terrain that prevent units from moving at their maximum speed. Additionally, we introduce a new initial distribution of units.

The line that marks the center of the formation of units will start aligned along the x-axis with its center positioned at $(-27, 0)$. Its extent is still 44. After 50 seconds, the line will move along the x-axis until its centerpoint is located at $(20, 0)$ with a speed of 0.25 per simulated second. After it has been at rest for 30 seconds in this position, it will move towards $(20, 40)$ and at that point, the simulation ends.

Units are placed initially in the lower left part of the environment. Once 150 seconds have passed in the simulation, two new units will be added in the lower

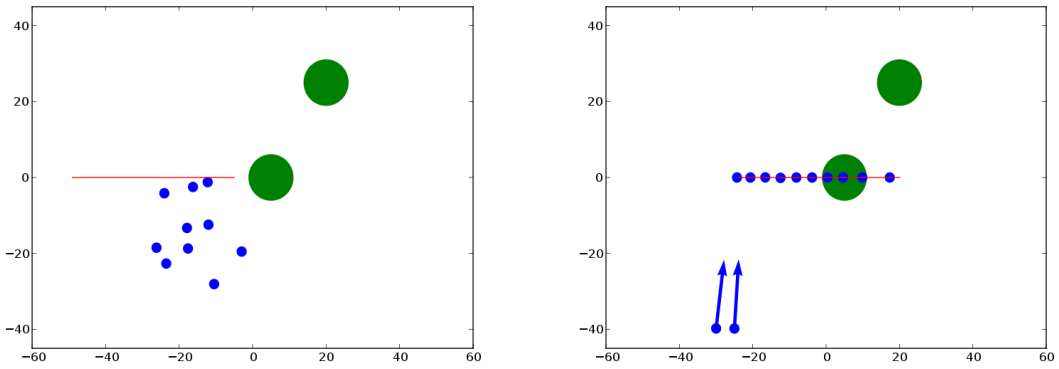

Figure 4.12: Initial distribution of units for robustness test of line and distribution when new units are added.

left of the environment. These should join up with the line formation. Units have maximum speeds picked from $\{0.3, 0.35, 0.4\}$ such that every third unit has each of these maximum speeds.

At $(5, 0)$ and $(20, 25)$ there are two circular terrain patches of radius 6 that prevent units from moving faster than 0.2. The line moves through these patches. The entire line moves though the first of these patches and an extent of the line formation moves through the second one.

No optimization or tuning of parameters will be performed for this experiment, only parameters that were found in the line experiment will be used.

Performance Measure

The performance measure is the same as for the line experiment. It is reproduced here for convenience:

$$p(t) = \frac{1}{n(t)} \sum_{i<n(t)} r(i,t)$$

$$P(S) = 30 \cdot d(S) + \frac{1}{T(S)} \sum_{t<T(S)} p(t)$$

This is the average distance from the units to the line plus a penalty for each collision. For a more thorough explanation, see 4.1.

Technique	c_2	σ_2	c_1	σ_2	$P(S)$
Handcrafted	-0.5	-0.5	7	2.3	2.58148
ES	-0.666186	-0.480902	9.60076	2.62909	2.51007
SA	-0.409996	-0.333396	10.2848	3	2.52954
PSO	-0.992032	-0.334872	14	2.87928	62.5073
GA	-0.8736	-0.960802	10.4899	2.33163	62.2868

Table 4.16: Robustness of discovered paremeters from the line experiment.

Solution strategy

The same solution strategy that was developed in 4.1 will be used here. The potential function that will be applied is:

$$a(i) = \frac{-c_2}{r_l^{\sigma_2}(i)}$$
$$f(i,j) = \frac{c_1}{r^{\sigma_1}(i,j)}$$
$$\mathbf{F}(i) = \vec{u}_l(i)a(i) + \sum_{j \neq i} f(i,j)\vec{u}(i,j)$$

This potential function will be tested with the best parameter values that were found by each optimization technique and trial and error in the line experiment.

Results

Table 4.16 shows the performance of the different parameters from the line experiment for this more complex environment. It is interesting to note that the two highest performing potential functions from the line experiment both cause units to collide in this environment. These parameter values performed well in the line experiment because they pulled units very close to the line.

Common for all of the experiment runs is that the units get squeezed together tightly while the line passes the first patch of terrain where they are forced to move more slowly. This is where the collisions happen. Units that are positioned on the line exert repulsion around them, so the visualization makes it seem that the line formation moves because the unit that is furthest to the back is pulled to the line and it exerts a push on the units ahead of it. When the units pack too closely together, this push causes one unit to dart into a unit that has been slowed down ahead of it.

All parameters result in coherent line formations. Not all units are able to

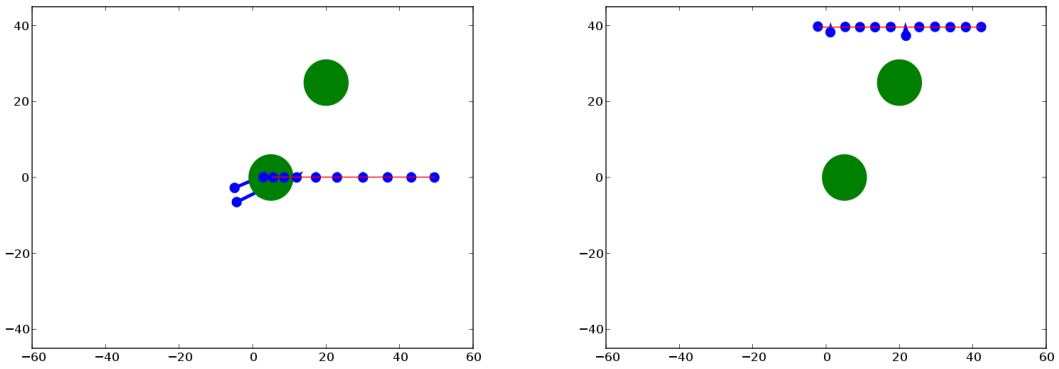

Figure 4.13: Two units about to collide in the tail of the line formation and the best final configuration found.

catch up with the line after passing through the second patch of terrain where they move slower. The two units that spawn at a later point in time only catch up with it during the last few seconds.

Conclusion

Clearly changing the environment of the experiment was too much for some of the potential functions that were discovered. However on the whole, the results strongly indicate that it is possible to train potential functions in one environment and utilize them in others. The case with 3 out of 5 trials was that all units survived and kept to a coherent line formation in a very satisfactory manner.

Robustness of box formation

Objective

Discover whether experimental results from previous experiments can scale as the number of units in the formation increases.

Environment

No obstacles. Units have a simulated friction to limit their speed. Their velocity is calculated by adding acceleration to it at every timestep. Their acceleration is calculated from potential fields discovered in 4.1. Units are not allowed to collide.

Motivation

The utility of the box formation setup would increase significantly if it were to be able to scale to accomodate larger numbers of units.

Performance Measure

The average area of the smallest square that can fully contain all units for each timestep divided by the amount of space occupied by units at that timestep.

Expected Outcome

The potential functions found in the box formation experiment are optimized to pack units quickly into a small area. Therefore, we do not expect to be able to add many additional units before collisions happen. We also expect the shape of the formation to become more disc-like as more attraction is added towards the center of mass of the formation.

Description

This environment is identical to the one used in the box experiment, with a minor modification - units have radius 1.5 instead of 2. The same initial distribution of units is used. This should enable the addition of several units before the first collision, allowing inspection of the shape of the formation as it grows.

Units are added from the positions $(-40, -40), (40, -40), (-40, 40), (40, 40)$ every 45 seconds in the simulation. This will grow the bounding square and the new units will attempt to join up with the formation. In doing this, they will create repulsion towards its center. It is only a matter of time before the repulsion towards the center becomes so strong that units collide. When the first collision happens, the simulation terminates.

Performance Measure

The same performance measure that was used for the box experiment will be used. It is repeated here for convenience:

$$p(t) = \frac{A(t) + 225d(t)}{n(t)4\pi}$$
$$P(S) = \frac{1}{T(S)} \sum_{t<T(S)} p(t)$$

For a thorough explanation of the meaning of these terms, see 4.1. As one of the objectives here is that the shape of the formation stays close to a box formation, visual inspection will also be used to discuss the results.

Solution strategy

The same solution strategy that was used in the box formation experiment will be used here. The test will be run with the best potential functions found by PSO, ES, GA, SA and trial and error.

Results

Table 4.17 documents the performance of the different weights that were applied to this experiment. The row A denotes the amount of units that were alive in the simulation prior to the first collision.

Parameters	ES	PSO	GA	SA	Handpicked
c_1	-0.001	-1.12931	-0.936459	-0.069546	-1
σ_1	-1.38582	0.173039	0.285367	-0.369282	1
c_2	9.00403	17.3137	14.123	11.7752	9
σ_2	2.34396	1.31857	1.4454	1.83189	2
c_3	-0.001	-0.532291	-0.37234	-0.377307	-0.0375
σ_3	-0.001	-0.378826	-0.294172	-0.381649	-0.05
A	40	44	40	40	24
$P(S)$	9.55571	5.28116	8.27221	8.53572	18.6356

Table 4.17: Performance of parameters from the box experiment in expanding its size to the maximum number of units before a collision. A is the amount of units in the formation.

It is a little surprising to see how much better the optimized potential functions perform on this experiment than the hand picked one. Clearly their characteristics

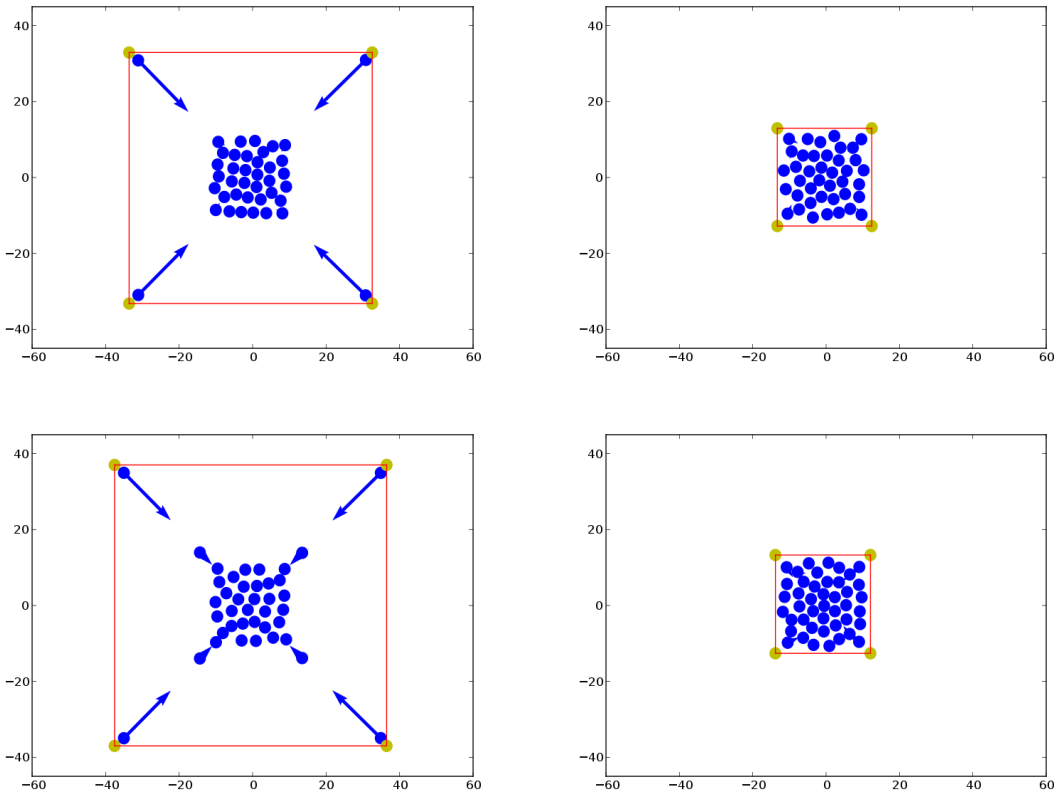

Figure 4.14: Last simulation frame before collision for SA, GA, ES and PSO (left to right, top to bottom).

are better for this test. They all use higher repulsion for the intergroup force law than the human set, with the exception of the ES parameters, which uses much less intergroup attraction. Interestingly, the PSO parameters that were not best for the box experiment itself performs much better here than the alternatives.

Figure 4.14 documents the last simulation frame of each of the optimized potential functions. The shapes of these box formations are all very close to square. This is in part because units at the corners are pulled, but also in part because that's where units join the formation. The shapes become more disc-like between the intervals when new units join the formation and the virtual leaders in the corners are pulled out of their influence range.

Conclusion

None of the potential functions that were found scaled as well as hoped for. The maximum amount of units in any formation before there were collisions was 44. That is a reasonable number, but because there is lag in initiating movement of the formation, it would decrease significantly for a moving formation.

As Figure 4.14 shows, the density of units near the middle of the formation is very high. This is because pull from the virtual leaders at the corners of the box doesn't reach this area at all. Clearly, a different approach for the potential function from the edges of the box is needed. Adjusting the range of the attraction from the virtual leaders up to half the diagonal of the box will ensure that there is always some pull here. However, this introduces another problem - these attractive zones will overlap and create deadzones, where no units will go. Essentially this leads to 5 areas of densely packed units near the corners and in the middle of the formation.

Were it not for this problem, it could be concluded that different potential functions with the same setup of virtual leaders could be found that would enable the formation to contain any reasonable number of units. However, this is not the case, because the problem of densely packed units in the middle of the formation is caused by this area having no attraction to corners. This becomes very visible when units join the formation - the bounding square grows such that the virtual leaders are pulled out of range from the middle of the formation, which leads to denser, disc-like packing.

Robustness of wedge formation

Objective

Discover whether experimental results from previous experiments can be applied to more complex environments.

Environment

The environment has patches of terrain where units move more slowly. Units can move at a limited speed that may differ between units. Their velocity is calculated by adding acceleration to it at every timestep. Their acceleration is calculated from potential fields found in 4.1.

Motivation

The utility of potential fields would be much greater if it is possible to use simple environments to create formations that work in a vide variety of environments.

Performance Measure

Minimize the amount of units that collide and maximize the amount of units in a predefined triangle shape that moves.

Expected Outcome

Because the potential setups discovered in 4.1 are optimized to pack a wedge tightly and quickly, it is expected that this environment that requires units to space out more in order to be safe, will be very challenging and many collisions are expected.

Description

This experiment takes place in a world that is similar to the one used for the wedge experiment. Units now have limited speeds, picked from the set $\{0.4, 0.45, 0.5\}$ in turn. There are initially 15 units, left part of the simulated world. Once 150 seconds have passed, 3 more units will be added in the upper left of the world. These should join up with the wedge formation.

The triangle that defines the wedge formation is initially placed with its corners at $(-50, -40), (-50, 40), (-10, 0)$. When it moves, it moves with a speed of 0.15 per simulated second. Letting seconds be the amount of simulated time that has passed and direction be a two-dimensional vector that will be normalized and multiplied with the speed of the line, its movement can be stated as:

```
if 90 < seconds < 260:
   direction = {1, 0}
if 260 < seconds < 440:
   direction = {1, 1}
if 440 < seconds < 460
```

```
    direction = {1, −1}
if 600 < seconds < 780
    direction = {−1, 0}
```

There are patches of terrain that slow units down to 0.1 in this environment. They are circular and have a radius of 4. They are located in the following locations: $(0, 0), (10, -5), (20, 10), (-5, -10), (-10, 15), (10, 20$.

The simulation terminates after 800 seconds have passed.

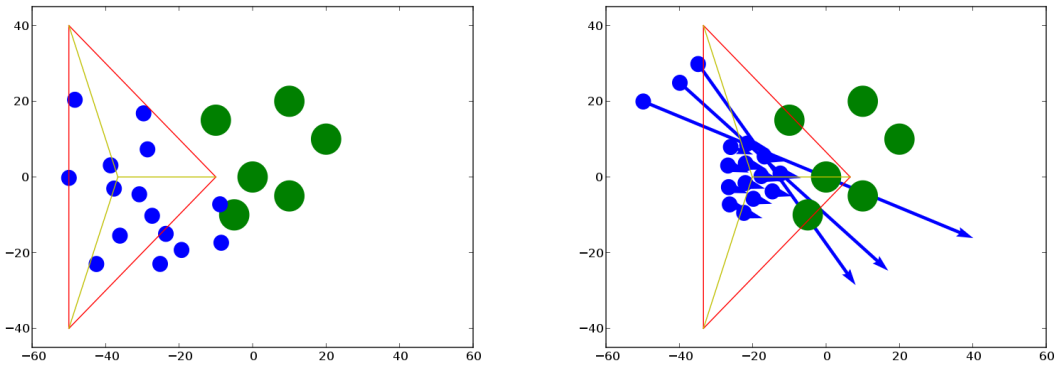

Figure 4.15: On the left, initial distribution of units for wedge robustness test. On right, three new units appearing and trying to join formation.

Performance Measure

This simulation uses the same performance measure as the one developed in 4.1 with a minor modification. The performance measure is repeated here for convenience:

$$p(t) = \frac{a(t)}{n}$$

The modification we require is to let n become $n(t)$ - the maximum amount of units that could be alive at timestep t. This is because we add units to the simulation while it is running.

$$P(S) = \frac{1}{T(S)} \sum_{t<T(S)} p(t)$$

For a more thorough explanation of this performance measure, see 4.1.

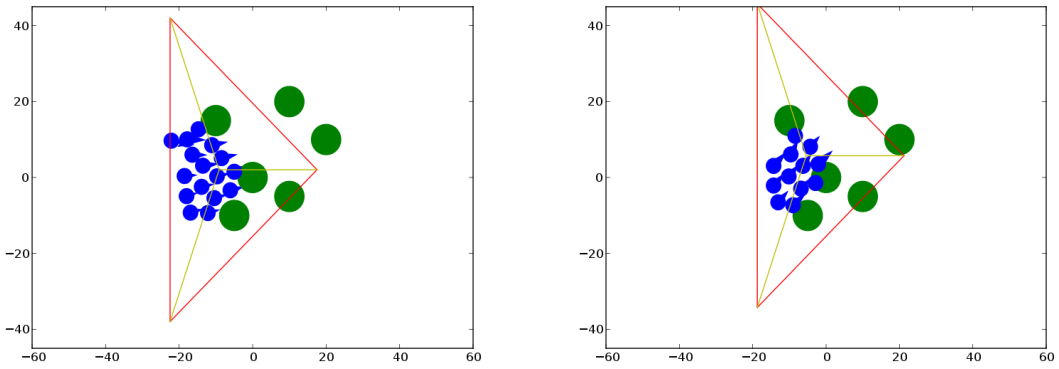

Figure 4.16: Two visualizations of the simulated annealing parameters used, right before and right after collisions.

Solution strategy

The same solution strategy that was used in the wedge experiment will be applied here. Letting $\vec{u}(i,j)$ give the direction from unit i to unit j and $\vec{u}(i,v)$ give the direction from unit i to the virtual leader v, the potential function is:

$$\mathbf{F}(i) = \sum_{j \neq i} \left(\vec{u}(i,j) \frac{c_1}{r^{\sigma_1}(i,j)} \right) + \sum_{v} \vec{u}(i,v) \left(\frac{c_2}{r^{\sigma_2}(i,v)} \right)$$

Results

Table 4.18 documents the performance of the different weights that were found for the wedge experiment, applied to this more complex environment. It is clear that none of them perform particularly well. Most of the weights do manage to gather all the units into a coherent wedge formation, but collisions take place almost as soon as the formation starts moving. In some cases, units that are very far away from the formation get pulled so strongly towards it that it dominates repulsion between them even when they have almost collided.

Because the performance metric for the wedge experiment was focused on quickly packing units into a wedge, the wedge formations become very tight and offer very little space for manouvering. This becomes a problem when units move at different speeds and it is exacerbated by the areas where units are forced to move slowly. Most collisions take place when the center of the formation is in the middle of the terrain patches that slow down units.

To exclude the possibility that the premises for this test were faulty, some further testing was performed that discovered a simple potential function that allows

Technique	c_1	σ_1	c_2	σ_2	$P(S)$	Survivors
Handpicked	9	1	-0.5	-1	0.392049	0
GA	12.7609	1.08366	-0.696471	-0.914919	0.326027	0
SA	4.57825	1.22634	-0.183973	-0.929281	0.360501	0
ES	14.3339	1.21731	-0.520555	-0.907055	0.654116	6
PSO	12.8845	0.856295	-0.892512	-1.02084	0.342038	0
Custom	20	1.25	-0.5	-0.5	0.7574	18

Table 4.18: Wedge robustness test results.

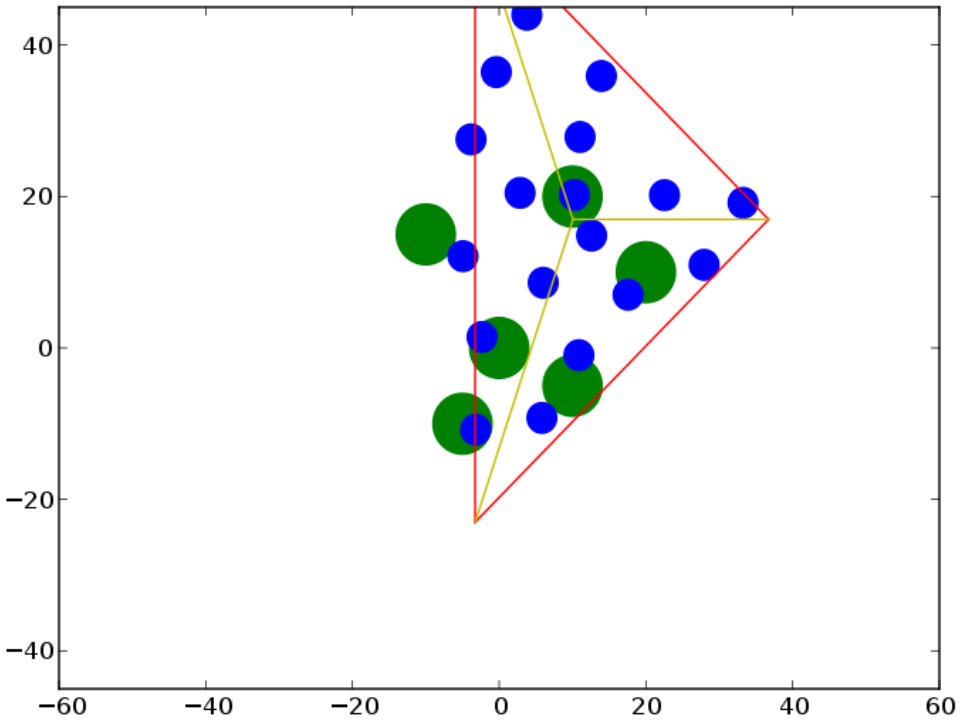

Figure 4.17: Potential function that does well on the wedge robustness test - the Custom entry in Table 4.18.

more units to live and a more coherent wedge formation to manouver the terrain. This is documented in Table 4.18 as the entry named Custom. The formation this potential function creates is nowhere near as tightly packed as the other ones tested - with good reason. The wedge shape is much larger than it needs to be to accomodate this amount of units and the penalty for collisions is much higher than the penalty for straying outside of the triangle.

Conclusion

We can conclude that none of the potential functions discovered in 4.1 transfer particularly well to the environment used here. Because this environment is not highly complex compared to those found in many real-time strategy games, it is clearly not sufficient to simulate a static wedge formation to discover potential field functions for one that needs to move through a difficult environment. In all cases, the repulsion between units was insufficient to save them from collisions when their speeds are different and they get clumped heavily because of terrain.

This is a result of three different factors:

- The environment in which the wedge shape was trained is insufficiently complex to transfer the shape to the environment in this experiment.
- The performance measure focuses on packing into the triangle quickly, which leads to results that apply as much pull as possible to the center of the formation.
- The wedge shape was trained with units that were initially distributed in a very different way from the distribution used in this experiment.

The greatest contributor here is the design of the performance measure. We conclude that optimizing the potential functions for safety over speed may improve these results considerably, but rather than exploring that further, propose this hypothesis as future research.

Chapter 5

Evalution and Conclusion

5.1 Simulation environments and real-time strategy games

Rationale for creating a simulation environment

The simulation environment used for this research differs from typical real-time strategy games in some ways. While an existing game could have been used for the simulations, this limits the type of environment that could have been simulated. Furthermore, existing methods of hooking into well-known games present significant overhead that makes it impractical to run many thousands of games in a short period of time. Other games could have been used that permit hooking directly into the game engine with AI, such that the more time-consuming parts of the game could be turned off to permit more games to be run. This however does present the problem that a complete agent needs to be prepared, complete with planning, path-finding and understanding of the internal data types used in the game.

A simulation environment avoids these problems and allows for much more testing to be done in a shorter timeframe. As long as the simulation environment can be made reasonably similar to a game environment, it should be possible to import potential fields found in the simulation environment to the game. The game engine part of this can be made less complex, as learning can happen in the simulation system.

Comparison of simulation environments and RTS games

Real-time strategy games typically do not use contineous coordinate systems, such as the simulation environments use. Instead, they have some smallest discrete unit of size, often a pixel. Units occupy areas corresponding to some discrete amount of this size. Terrain is created by setting up tiles that contain some discrete amount of this size. A tile can contain pixels where units can not walk, these are obstacles. It may also contain terrain with different properties, areas where units walk more slowly or similar. Commonly, a tile is either an obstacle or it is not.

This difference between the simulation environments and real-time strategy games is not a big hurdle. It seems easier to adapt potential functions created for contineous environments to discrete environments than the other way around. Concepts such as euclidian distances, directions and movement speeds can still work the same way.

While units in most real-time strategy games are not circular, these games operate with the concept of a “hitbox”. The hitbox is the actual area occupied by the unit, regardless of its visual shape. It is used to calculate distances, collision detection and similar. It is commonly rectangular. This is a minor difference, units in the simulation system could have had more complex shapes, but this does not seem necessary in order to test the method.

Collisions in real-time strategy games do not typically result in unit death or even damage. Instead, they have solid body physics, such that units simply are not able to move “into” one another. The simulation environments used harsh penalties for collision to ensure that units would be forced to spread out over an area instead of gathering in a clump. It is possible that a better approximation to a game would be to disallow units moving into one another and leave it at that. However because the shape of a formation is important, we have chosen to penalize formations that have areas where units clump very tightly. It is also unclear what would happen if say 3-4 units attempted to move to the same exact location in a game.

While some real-time strategy games do operate with concepts of acceleration, they often simplify it greatly. For example, units may have 3 different velocity magnitudes to choose from. They are either at rest or travel with one of these velocities. This leads to units that can stop instantly. The simulation environment used acceleration and deceleration. It may be that a closer approximation to a real-time strategy game would be to use immediate actuation, that is to set the velocity directly from the forces calculated. It also seems clear however that if potential field based troop movement can be made to work in the more complex case where acceleration is handled, they can be made to work with velocities.

Overall, many of the properties of the simulation environments approximate

the real world better than the typical real-time strategy game does. In particular, the presence of acceleration and deceleration is a property that makes it more complex to develop good control mechanisms. As noted in [Laird and VanLent, 2001], [Buro and Furtak, 2003], realistic games provide a very interesting testbed for AI methods. The simulation environments are made more suitable by the presence of more realistic physics.

5.2 Summary of results

Line formation results

The potential field setup for line formations discussed in this thesis creates well-behaved groups of units that closely resemble a line formation. The results indicate that several optimization techniques are able to create potential functions that accomplish this. The potential functions created for the line formation perform well in a more complicated environment and the line formation is able to accomodate additional units with ease in most cases. Units in the formation adapt well under difficult circumstances, such as when the line moves faster than they are able to or when they have non-uniform maximum speeds.

Box formation results

Box formations are created that pack units quickly into a shape that strongly resembles a box along its edges. The box formation packs more tightly in its center than in its outer regions and does not scale up to accomodate additional units gracefully. Applying optimization to the box can result in potential functions that create boxes that pack quicker and tighter. When units are added to the box formation, the bounding square increases in size and the virtual leaders in the corners move such that their minimum range of influence is too short to reach the middle of the formation. This makes the box formation become more disc-like. Due to the tight packing in the box, it it not particularly well-suited for moving formations.

Wedge formation results

Potential field functions that create wedge-shaped formations are created with relative ease. The edges of the formation screate the desired v-shape with a heavy weight of units behind the tip of the formation. Like the box formation, the wedge formation packs units more tightly in its center, which is desirable to give the wedge punching power. Optimization results in wedges that pack quickly and tightly. Applying the discovered wedge formations to more complex environments results in several collisions. Despite the large wedge shape the units are allowed

to fill, the potential functions create very tightly packed wedges that are ill-suited for movement through difficult terrain.

5.3 Conclusion

Created formations

We have shown how potential field methods may be applied in order to create unit movements that closely match classical line, box and wedge formations in environments with rules similar to those found in real-time strategy games.

While the internal structure of the box formation discovered does not match the ranks-based or hexagonal grid-based structures often seen in this type of formation, it is clearly possible to create an external structure that is close to this. The wedge formation is easily and efficiencly created using the box formation as a basis and it closely resembles traditional wedge formations.

We can conclude that potential field based methods can create classical military formations. Because the potential field setups can be created with a high level of abstraction per formation type, they are well-suited for the purpose of *gathering* units into formations with classical outward shape. For wedge formations, they also create a desirable internal structure and shape.

Transferrability of created formations

We have shown that it is possible to create potential field managed line formations that function well in more diverse environments than they were developed in. The line formation can accomodate different amounts of units than it was created for with very little difficulty and it also works as expected using units with non-uniform maximum speed. This result strongly indicates that it is possible to use potential field methods to create group behaviour that transfers well between different environments and groups of units.

The box and wedge formations that were developed do not transfer well. It is not possible to conclude that potential field methods in general can not achieve this objective because the performance measures used for the environment were too focused on properties of the formation that directly work against transferrability. Notably, they create formations where units pack quickly and tightly, which means that the amount of space in which to manouver inside the formation is minimal. More research is needed to establish whether these formations can be managed well by potential field methods. Using different performance measures, adjusting the solution strategies with different potential functions or introducing

an additional parameter for the amount of units in the formation to the potential functions are promising ideas.

Suitability for learning

Offline learning techniques are well suited for finding potential functions that result in desired behaviour, if that behaviour can be specified accurately with a numerical fitness function. This is clear from the fact that all optimization techniques applied in the experiments were able to find satisfactory potential functions in all cases. The lack of transferrability of the functions found can in very large part be attributed to performance measures that do not accurately reflect behaviour that transfers well.

For cases where it is possible to evaluate several sets of potential functions in parallell, PSO and GA are great techniques for optimizing potential functions. Simulated annealing with the parameters used in this thesis did not deliver good results *reliably*. Evolutionary strategies have many of the same characteristics as genetic algorithms, but performed worse for this thesis. The genetic algorithm created to work with a vector of reals genotype performs remarkably well, despite being run with a lower population size than what is common for this type of algorithm. Particle swarm optimization also delivers reliably with an even lower population size. It is however clear that all the optimization techniques employed for the experiments can produce satisfactory potential functions. This strongly indicates that the problem is well suited for learning, especially when taking into consideration the relatively low numbers of evaluations that were used for training. Arriving at feasible results with low numbers of evaluations is naturally a very favorable property - evaluating performance in a real-time strategy game is not computationally inexpensive.

In the case with the box formation, learned solutions outperformed solutions that were created by hand with a wide margin. For the other formation experiments, the gain was less significant. It is unclear whether the improvement is worth the configuration time and computation time for these formations, but these techniques can be employed to efficiently explore possibilities for potential functions. Furthermore, the indication that potential fields can be learned as a control mechanism increases the utility of the method for being applied to different environments.

Overall conclusion

More research is needed to conclude whether potential field methods constitute a good control mechanism for groups of units in real-time strategy games. However the results in this thesis strongly indicate that the method has the potential to

fullfil this property.

This thesis applies offline learning techniques to potential field methods and shows that it is possible to combine these to create a control mechanism for groups of units in real-time strategy games that is applicable for learning. More research is needed to establish whether the method is well suited for online learning techniques.

5.4 Future work

Finding box and wedge formation setups that transfer well

The box and wedge formations documented in this thesis do not transfer well to more complicated environments and different amount of units. Being able to find setups that work for more units, in more complicated environments is necessary to use potential field methods for real-time strategy games.

Some possibilites for this include: looking into nearest-neighbours potential functions, using repulsive edges on bounding box of a square function, with attractors in the corners and repulsive units, using distance of influence potential functions for repulsion amongst units, training the formations in more complex environments and building potential functions that take as input the amount of units in a formation.

Transfer learning approaches may be worth investigating to accomplish this goal. The problem of making our box and wedge formations work in more diverse environments and with different groups of units is very similar to the problems investigated in [Wilson et al., 2008] and [Sharma et al., 2007].

Degree of complexity required for training situations

One reason why the line formation from this thesis transfers better than the box and wedge formations is because it was created in a more complex training situation than the box and wedge formations were. Naturally it is therefore a more useful result. However, specifying a more complex training situation requires more work. Therefore it would be useful to know the degree of complexity that is required for behaviour to transfer from training situations to other environments.

Using learning to construct complete potential functions

This thesis concerns itself only with optimization of parameters for predefined potential functions. It would be very interesting indeed to apply optimization to create potential functions that could be functions of more types of input such as current velocity or current acceleration. A learning system could be developed that can train the potential field setup in full, such that it attempts different types of virtual leader layout for formations. This could possibly be achieved with a genetic programming approach [Koza and Poli, 2005]. Another way would be to allow the training system to encode the virtual leader layout as a part of the genotype, allowing it to pick potential functions to evolve from a predefined set of functions.

Learning from multiple environments for one formation

It seems likely that one way to increase the transferrability of potential field setups would be to learn in multiple environments, instead of a single, simple one. It would be interesting to observe how this would affect results and it would be interesting to find out how complex environments need to be to produce good potential functions for a game.

Implement rts-game behaviour for units

The simulation in this thesis does not have any particular behaviour for units other than as a group. In a real game, units have attacking and defending moves, and concepts such as attack range become important for finding good formations. For units with only melee attacks, keeping a strict formation makes sense. For a mixed formation of ranged and melee units, it may not. Testing should be performed for battle simulations where units have individual behaviour, such as attacking enemy units.

Calculating potential functions on the GPU

Being able to calculate on the GPU would make the method scale to much higher amounts of units in a simulation. In turn, this enables for more extensive training and testing. It may be that the reduction operation needed to sum up the forces on each object is too expensive for this to make sense, but calculating the force between any pair of objects becomes so cheap that this is worth trying.

Implementation in a real game

To truly discover the suitability for potential field methods for distributed behaviour in real-time strategy games, an implementation in an actual game is necessary. There are several candidates for this. While [BWAPI, 2012] allows for implementation for StarCraft: BroodWars, it seems that the method is a much better fit for environments where it can operate on the game engine level directly. BWAPI is created such that bots using it operate on the graphical user interface level. An interesting candidate is [Wildfire Games, 2012]. This is an open source game that is structurally similar to StarCraft, based on the popular Age of Empires franchise. The project is currently looking for programmers to implement the artificial intelligence for the game. Some work has been made in economy and planning but reportedly the micro management of the AI is not very strong: [Kogelnig, 2012]. The current AI uses potential fields to decide structure placement, so the technology seems like an excellent fit for this particular game. Another possible target environment is the ORTS platform, see [Buro, 2003].

Appendices

Folders and files included with thesis

Media

The media folder contains animated videos of experimental results for all 3 formation experiments and all 3 robustness tests. The folders contain animations of the best potential functions discovered for the experiment by each of the following optimization techinques: PSO, SA, GA, ES. Additionally it contains a video of the field generated by human settings. The wedge robustness folder has an additional animation, because an additional potential function was created for this experiment.

Figures

The figures folder contains all the fullsize figures created for this thesis. They are unfortunately not organized according to which section they have been used for, but they do have descriptive names.

Source

The source folder contains the source code of the simulation system that was created for this experiment. The README file here contains the installation instructions for the system. These are not very detailed and the installation procedure has not been tested first hand for other systems than linux_x64. It may be that the Cython (ccalcs.c) generated C code that is included with this distribution is not compatible with 32-bit platforms. Running the cython command-line command on ccalcs.pyx should create a new, compatible version if this is the case. This file is located in source/apf-0.1.3/apf/lib/ccalcs.pyx.

Experiments

This folder contains the input data and scripts that were written to perform the experiments for this thesis. Each folder has at least the following files:

apfrc
: The settings to use for running the experiment through the simulation system.

indata.json
: Initial, statically placed units. Initial force laws if any.

script.py
: The script that was used to run the experiment. In general this contains functionality for running the experiment with different parameters, visualizing it live, plotting to files or generating videos. May additionally contain code to hook experiment up with optimization techniques in the inspyred library.

Experiments that were optimized contain a number of csv-files in a folder named csvs. These are named according to which optimization type they detail, which initial random seed that was used and whether they contain detailed data about individuals. Inspyred has an analysis module that can digest these files to produce fitness plots and similar statistics.

References

Barnes, J. and Hut, P. (1986). A hierarchical 0 (N log iV) force-calculation algorithm. *nature*, 324(4).

Bell, G. (2005). *Forward Chaining for Potential Field Based Navigation*. PhD thesis, University of St. Andrews.

Bell, G. and Weir, M. (2004). Forward chaining for robot and agent navigation using potential fields. In *Proceedings of the 27th Australasian conference on Computer science-Volume 26*, pages 265–274. Australian Computer Society, Inc.

Buro, M. (2003). ORTS: A hack-free RTS game environment. *Computers and Games*, pages 280–291.

Buro, M. and Furtak, T. (2003). RTS games as test-bed for real-time AI research. In *Proceedings of the 7th Joint Conference on Information Science (JCIS 2003)*, pages 481–484.

Buro, M. and Furtak, T. (2004). RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, volume 6370.

BWAPI (2012). BWAPI. `http://code.google.com/p/bwapi/`, last accessed May 2012.

Deb, K. and Padhye, N. (2010). Development of efficient particle swarm optimizers by using concepts from evolutionary algorithms. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 55–62. ACM.

Floreano, D. and Mattiussi, C. (2008). *Bio-Inspired Artificial Intelligence*. MIT Press.

Ge, S. and Cui, Y. (2000). New potential functions for mobile robot path planning. *Robotics and Automation, IEEE Transactions on*, 16(5):615–620.

Ge, S. and Cui, Y. (2002). Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13(3):207–222.

Hagelbäck, J. and Johansson, S. (2008). The rise of potential fields in real time strategy bots. *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Hagelbäck, J. and Johansson, S. (2009). A Multi-agent Potential Field based bot for a Full RTS Game Scenario. *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Huang, H. (2011). Skynet meets the Swarm: how the Berkely Overmind won the 2010 StarCraft AI competition. `http://arstechnica.com/gaming/2011/01/skynet-meets-the-swarm-how-the-berkeley-overmind-won-the-2010-starcraft-ai-competition/`.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220:671–680.

Klein, D. (2010, accessed May 2012). The Berkely Overmind Project. `http://overmind.cs.berkeley.edu/`.

Kogelnig, R. (2012). AI in 0. A. D. `http://aigamedev.com/open/interview/ai-in-0ad/`, last accesses June 2012.

Koren, Y. and Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404. IEEE.

Koza, J. and Poli, R. (2005). Genetic programming. *Search Methodologies*, pages 127–164.

Laird, J. and VanLent, M. (2001). Human-level AI's killer application: Interactive computer games. *AI magazine*, 22(2):15.

Leonard, N. and Fiorelli, E. (2001). Virtual leaders, artificial potentials and coordinated control of groups. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, volume 3, pages 2968–2973. IEEE.

Long, E. (2007). Enhanced NPC behaviour using goal oriented action planning.

Nurmela, K. and Östergård, P. (1997). Packing up to 50 equal circles in a square. *Discrete & Computational Geometry*, 18(1):111–120.

Orkin, J. (2004). Symbolic representation of game world state: Toward real-time planning in games. In *Proceedings of the AAAI Workshop on Challenges in Game Artificial Intelligence*.

Perkins, L. (2010). Terrain Analysis in Real-Time Strategy Games: An Integrated Approach to Choke Point Detection and Region Decomposition. In *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Reif, J. and Wang, H. (1999). Social potential fields: A distributed behavioral control for autonomous robots. *Robotics and Autonomous Systems*, 27(3):171–194.

Reynolds, C. (1987). Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34. ACM.

Rimon, E. and Koditschek, D. (1992). Exact robot navigation using artificial potential functions. *Robotics and Automation, IEEE Transactions on*, 8(5):501–518.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice hall.

Sharma, M., Holmes, M., Santamaria, J., Irani, A., Isbell, C., and Ram, A. (2007). Transfer learning in real-time strategy games using hybrid CBR/RL. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, number 1041-1046.

Vadakkepat, P., Lee, T., and Xin, L. (2001). Application of evolutionary artificial potential field in robot soccer system. In *IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th*, pages 2781–2785. IEEE.

Vadakkepat, P., Tan, K., and Ming-Liang, W. (2000). Evolutionary artificial potential fields and their application in real time robot path planning. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 256–263. IEEE.

Weber, B. (2010, accessed September 2011). Aiide 2010 starcraft ai competition. `http://eis-blog.ucsc.edu/2009/11/aiide-2010-starcraft-ai-competition/`.

Weber, B. (2012). *Integrating learning in a multi-scale agent*. PhD thesis, University of California, Santa Cruz.

Weber, B., Mateas, M., and Jhala, A. (2011). Building human-level ai for real-time strategy games. In *2011 AAAI Fall Symposium Series*.

Weber, B. G., Mawhorter, P., Mateas, M., and Jhala, A. (2010). Reactive planning idioms for multi-scale game AI. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 115–122. IEEE.

Wildfire Games (2012). 0 A.D. `http://wildfiregames.com/0ad/`, last accessed June 2012.

Williams, D. (2002). Structure and competition in the US home video game industry. *International Journal on Media Management*, 4(1):41–54.

Wilson, A., Fern, A., Ray, S., and Tadepalli, P. (2008). Learning and Transferring Roles in Multi-Agent Reinforcement.