High-Level Control of Agent-based Crowds by means of General Constraints

A dissertation submitted in satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

David Jacka

February 2009

I know the meaning of plagiarism and declare
that all of the work in this document, save for that
which is properly achnowledged, is my own.

for the Lord Jesus, creator of what we can only simulate.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I must firstly acknowledge the huge contribution of my supervisors, James Gain and Patrick Marais, to this work. Thank you for your guidance, insight and hard work in helping me with this dissertation. I have learnt a great deal from you these past two years.

I must also thank the members of the Geometry Interest Group and my other lab mates, most especially Andrew, Ashley, Ian, my brother Jannie, Simon, Rudy, Bertus, James, JP, Chong and all the rest who were ready to join in a game of DOTA when we should have been working. You've also taught and helped me and have made working in the lab a whole lot of fun.

The National Research Foundation provided funding for my research in the form of a Sparce Skills and Grant-holder bursary, as did the Postgraduate Funding Office of UCT through the Manuel and Luby Washkansky, KW Johnston and Harry Crossley scholarships.

I must thank my parents for the love, support and financial backing all through my studies (as well as allowing me to raid your fridge).

To my lovely wife, Carryn, who has put up with a husband who spends long hours on "nerd stuff": I love you, thank you for your patience and support.

Finally, and most importantly, I thank God who gives me the ability to do this research. After all, there is more to life than playing with computers.

Abstract of the Dissertation

High-Level Control of Agent-based Crowds by means of General Constraints

The use of virtual crowds in visual effects has grown tremendously since the warring armies of virtual orcs and elves were seen in *The Lord of the Rings*. These crowds are generated by agent-based simulations, where each agent has the ability to reason and act for itself. This autonomy is effective at automatically producing realistic, complex group behaviour but leads to problems in controlling the crowds. Due to interaction between crowd members, the link between the behaviour of the individual and that of the whole crowd is not obvious. The control of a crowd's behaviour is, therefore, time consuming and frustrating, as manually editing the behaviour of individuals is often the only control approach available. This problem of control has not been widely addressed in crowd simulation research.

We propose, implement and test a system in which a user may control the behaviour of a crowd by means of general constraints. This Constraint Satisfaction system automatically alters the behaviour of the individuals in the crowd such that the group behaviour meets the provided constraints. We test this system on a number of scenarios involving different types of agents and compare the effectiveness of this automatic system to an expert user manually changing the crowd. We find our method of control, in most cases, to be at least as effective as the expert user.

CHAPTER 1

Introduction

The film adaptation of the epic *Lord of the Rings* trilogy brought to our screens vast armies of fantastic creatures battling for Middle Earth. The scale and success of these films is partly due to the use of these huge, computer-generated crowds (see figure 1.1). The software that created these crowds, *Massive*, has changed the way crowds are used in film. What was once an impossible expense even in large productions is now possible even for short films and commercials. Carling Lager recently ran an advert called "Big Ad" in which they made fun of the resulting trend of employing large scale crowds in film productions[1].

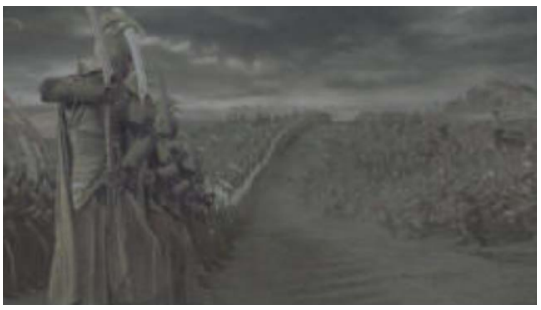


Figure 1.1: *A shot from the prologue of the first Lord of the Rings movie. Only the characters in the immediate foreground are live actors, the rest were generated using Massive. Image source: [ABL04]*

Computer generated crowds are realistic and believable, both in their low-level movement and in their high-level behaviour. This large scale complex crowd behaviour would be difficult to define in a holistic, top-down manner. Instead, software, such as Massive, animates these crowds by endowing each individual agent in the crowd with the ability to animate itself. Each agent is given its own behavioural model, or "brain", and acts as it sees fit. This *behavioural animation* of the individuals equips each agent with: senses, giving an awareness of the environment and other agents; decision capabilities, allowing the agent to determine

[1]http://theinspirationroom.com/daily/2005/carlton-draught-beer-big-ad/

a reaction to the situation it finds itself in, and a body capable of a range of movement. The interaction of a number of these individuals creates complex, lifelike group behaviour from relatively simple building blocks.

As the individuals in a crowd simulation interact, we observe a phenomenon known as *emergent behaviour*. According to the emergence test described by Ronald *et al.* [RSC99], emergence occurs when the low level design, in this case an individual crowd member, has well understood behaviour but the behaviour of the whole elicits surprise (see section 3.5 for such a surprise). This is the case when the combination of the behaviours of the individuals in a crowd create global behaviour whose relation to the that of the individual is no longer obvious.

This emergence is desirable as, in reality, crowds show similar emergent behaviour. The disadvantage of this phenomenon, however, is the unpredictability that it introduces. The change in the behaviour model of a single agent acting alone may be fairly predictable to the person making the changes. However, take even a small group of these agents and allow them to interact and the complex behaviour that is created by their interactions changes the predictability of this mapping. When the behaviour of these agents is altered, the effect of this change on the crowd as a whole is not nearly as obvious. This makes tweaking and controlling the behaviour of the crowd, as a whole, much more difficult.

Visual effects, the area in which this form of crowd simulation has been so successfully applied, has a number of specific requirements, not least of which is *control*. The directors and animators that make use of crowd simulations require specific action and properties of the generated crowd in order to tell a story. A crowd simulation used in visual effects, in contrast to other areas of application such as virtual environments, must produce crowds that are not only believable, but *directable*.

This is the premise of our research into methods of controlling agent-based crowds. Reynolds, in his seminal paper on behavioural animation in which he animates flocks of birds, makes a note of this lack of control [Rey87]. The bird's behaviour, though interesting, can prove to be frustrating when the user has an exact goal in mind. Patterson, in his work on the robot crowds for the movie *Robots* also notices the need for continual tweaking and fine tuning of individuals in the simulation in order to have them perform as required [Pat06]. Massive, the industry standard in crowd animation software, also suffers from similar difficulties [Dun02].

As computer generated crowds are more widely used, crowd software such as

Massive has begun to include ready-made agents that may be used to generate a number of standard crowds. These crowds can quickly fill a stadium with cheering fans or a town with wandering pedestrians. Unfortunately, these packaged crowds are unlikely to exactly meet the needs of a particular production: agents may collide, wander in the wrong direction and spread themselves too far apart. There are few methods of control available to the user other than directly altering the behaviour of the ready-made agents. Since agent brains may be complex – made up of thousands of rules – this is a challenging task for a user unfamiliar with the particular agent.

This is the problem that this dissertation addresses: the control of agent-based crowds for visual effects. We apply this control in the form of user specified crowd-level constraints which are automatically met by altering the way in which the agents of the crowd think. We propose and implement a system for meeting such constraints based on the way in which modern crowd simulations, such as Massive, implement the brains of their agents. The *fuzzy logic* rules of these brains are based on the vagueness of human language and so we may control the agents by altering the way in which they understand their behavioural instructions. This allows a user to change the way in with a crowd of agents think, simply by specifying constraints on their resulting actions.

An example of such a constraint is requiring a group of agents to have crossed a line in the environment by a particular time in the simulation. The agents' understanding of their behavioural rules is automatically changed such that their decisions and actions result in them all crossing the line within the correct time frame. Our approach is aimed at meeting the constraints while producing a minimal change in the underlying behaviour of the crowd.

The scope of this work is the design and implementation of a constraint satisfaction system. We do not deal extensively with the specification of the constraints themselves though we demonstrate their power and refer to previous work in which their usefulness is shown. In addition, we propose control aimed specifically at visual effects applications with their particular requirements and restrictions - most importantly the need for specific control. This area is where virtual crowds have found greatest use and where additional control is needed.

Our contribution is a novel method of crowd-level constraint satisfaction. This is done by altering agents' brain parameters in a manner similar to an expert user performing the same task. Our method is, however, automatic and may be accelerated through parallelisation to perform such changes in less time than a human user requires. In addition, we reverse engineer the successful principles of the Massive crowd animation platform in order to build on its success. This step makes our work directly relevant to the visual effects community.

1.1 Structure

The structure of this dissertation is as follows. We first survey current work in crowd animation with particular reference to usefulness in visual effects in chapter 2. Of particular interest is the Massive crowd animation platform and the fuzzy logic on which its agent brains are built. These two topics are discussed separately in chapters 3 and 4.

We then discuss the design of our system of control in chapter 5 with reference to how such a system should be evaluated. The implementation of our test system is described in chapters 6 and 7. The testing details, results and discussion are included in chapter 8, followed by a conclusion discussing application, limitations and future directions of our work in chapter 9.

1.2 Terminology

Our area of concern is crowd animation. Since this is almost always accomplished through some form of simulation we use the terms crowd animation and crowd simulation interchangeably.

CHAPTER 2

Background

The research into crowd animation, specific to visual effects, is largely conducted by companies for in-house use. In other fields, however, a large amount of published research into crowd simulation has been conducted. Each of these fields brings with it inherent requirements and constraints and often the goal of the research is quite different from that required by visual effects. A crowd produced for a virtual environment, for example, must be simulated and rendered in real-time. This is not the case for film crowds which may be simulated off-line but require an extremely high degree of realism in the motion of the virtual characters.

A difficulty found when discussing crowd animation techniques is the general lack of comparison between these crowd systems in the relevant literature. Crowd techniques are often not compared to their predecessors and evaluation of a new technique is usually in the form of an accompanying video and informal discussion which provides little usable information. This trend is likely due to diversity of focus in crowd simulation research, as well as the difficulty in determining a means of comparison.

Before comparing crowd simulation systems, such a basis for comparison of the different ideas and techniques must be established. We start by determining the requirements of film with regard to crowd animation. These specific requirements may then be used to compare the various techniques available to some extent. We present the work, categorised by level of abstraction, and relate it to our own. It must be stressed from the outset that the aim of this dissertation is to develop the animation of crowds for visual effects and so we consider previous work from this perspective.

2.1 Requirements for Film

In order to evaluate a crowd simulation system, the area to which it is being applied must be considered. Our area of application - Film - has a number of distinct requirements. By looking at previous uses of crowd simulation in the film industry, as well as considering general descriptions of the making of recent films in which crowd simulations were used, it is possible to extract these important properties and requirements.

Stephen Regelous, the creator of the widely-used *Massive* crowd animation system, identifies two conflicting requirements of crowds in film [THL04]. The first is a high degree of *automation*. The crowds in a film need to be large, detailed and have wide variation. Animating such crowds on a geometric level, perhaps using a traditional keyframe approach, would be too tedious and take too long to be feasible. The reason crowd simulations are used is to automate the animation of these crowds to some extent.

In opposition to this requirement of automation is the need for complete *control* on the part of directors and animators. The story that a film tells may have very specific requirements for a crowd and so directors would like artistic freedom, unlimited by the technical aspects of simulating the crowd. Animators need to be able to control, at both a high and low level, the behaviour that a crowd animation system produces and change the crowd if it is not doing what is required. This requirement of control, at the level of the individuals that make up a crowd, is evident in the per-shot tweaking of the crowds in films such as *The Lord of the Rings* [Dun02], *Madagascar* [Ker05] and *Robots* [Pat06].

In a technical sense, a film is a series of camera shots which are strung together in order to tell the film's story. These shots are usually short, only a few seconds in length [Dun02]. What is somewhat surprising is that, in practise, it has been found that there is little need for shot to shot coherence in a crowd [THL04]. Although one might expect that a single crowd simulation would run for large parts of a scene and extend over a number of shots, directors prefer to change and tweak the crowd for each shot. This is perhaps less surprising when one considers that each shot is, in a sense, a separate entity with different shots often farmed out to different effects houses [Rob08].

Because a crowd will likely be altered from one shot to the next, a crowd simulation will seldom need to run for more than a few seconds. This has implications for the behaviour of the crowd in that long term planning is not needed. The director supplies the crowd's context. What is important is the crowd's immediate, reactive behaviour.

Because crowds are set up and tweaked for each shot, the process of tweaking and initialisation needs to be made efficient. Animators are not necessarily programmers and there is often a short feedback loop in which changes need to be made and approved. A computationally efficient crowd simulation is, therefore, beneficial as interactivity is useful in this authoring and tweaking stage. However, as the final crowd render in a film is usually generated off-line, this interactivity is more a usability issue than an absolute requirement.

When considering the wide use of CG crowds in modern movies, it is clear that the characters that make up a crowd can vary widely. These range from the cartoon-like robots seen in *Robots* to the fantastic but life-like armies in *The Lord of the Rings* to crowds of real people in *Vantage Point.* Since a crowd animation system needs to cater to such a wide range of characters and creatures, ease of authoring and the adaptability of the crowd model is of importance. The process of creating the correct crowd behaviour is a lengthy one, as seen in the development of the behaviour for the crowds in *Narnia*, which took some nine months to develop [WWB06]. *Authoring* is therefore an important consideration.

Whatever the nature of the crowd, the motion and believability of the characters in the crowd is paramount. A seamless blend is needed between the simulated crowd, often in the background, and other characters on screen. These characters may be hand animated, in the case of an animated movie, or be real actors from live footage. In both cases, the motion generated by the crowd simulation must stand up to comparison with the foreground characters. The great importance of motion quality is clearly seen in movies such as *Narnia* [WWB06] where the motion capture alone took six months. Industrial Light and Magic, in movies such as *Star Wars Episode 1: The Phantom Menace*, used only a simple particle system to control the crowd's movement [THL04]. Their primary focus, however, was on the quality of the animation that their system produced – the complexity of the crowd's behaviour was of secondary importance.

Although the quality of the motion of the individuals in a crowd cannot be compromised, the richness of the behaviour of individuals is also of importance. A major advantage of widely used crowd animation software such as Massive is that they allow for complicated crowd scenes with complex interactions and crowd behaviour such as fighting, falling and climbing over one another [Dun02, ABL04].

The properties and requirements discussed above give us a basis for examining a crowd simulation's usefulness in visual effects. We list the five distinct requirements (in approximate order of importance) that we will use to compare and discuss the work done in the field of crowd simulation:

- **Animation Quality** - The characters must fit into the film and move in a believable or plausible fashion. The crowd animation system must allow for this.

- **Control** - Since a film is telling a story, not just creating a virtual world, there is a need for complete control at both high and low levels.
- **Automation** - The purpose of a crowd simulation tool is to automate work that is too tedious to be done by hand.
- **Ease of authoring** - Crowds with a wide variety of characters must be created, initialised and tweaked for each shot, often by non-programmers.
- **Behavioural Complexity** - Simulated crowds which are able to perform a wide variety of actions are more useful to a director.
- **Computational Efficiency** - Low computational cost means that large crowds can be created while still allowing for interactive visualisation.

It is important to note that much of the work we discuss has been aimed at applications that differ significantly from film. These metrics are an attempt to define what is important for crowds in the visual effects industry in particular.

2.2 Crowds at Present

An widely-used and effective method of simulating a crowd of characters is using a *bottom-up* approach. In general, bottom-up refers to the modelling of low level elements in order to produce the high level effect. In the case of crowds, this means modelling the individuals that make up the crowd and then combining them in order to model the crowd itself. This is particularly true for visual effects applications as an individual representation of the crowd members is required.

The modelling of the individuals in a crowd is an example of what Reynolds calls *behavioural animation* [Rey87]. While traditional animation takes place on the level of geometry, behavioural animation takes place at a layer above the geometry – the behavioural level. By specifying how a character behaves, an animator creates a character model that is, on the geometric level, self-animating. Where a character rig might be thought of as giving a character a body and defining its possible movement, a behavioural model gives the character a brain so that it can move its own body.

Real crowds are made up of people or creatures that can sense, think, react and plan. Behavioural animation techniques model this with differing degrees of abstraction. Current crowd models can be categorised by their level and form of abstraction.

2.2.1 Crowds of Particles

Particle systems are widely used in computer graphics for modelling smoke, fire and water [Ree83]. Each particle in a particle system has a short list of attributes such as velocity and position. By adding orientation to this list of parameters we may use a particle as a simple description of an individual in a crowd. These particles are then subjected to forces which accelerate them through space - the particles themselves are inert. The use of particle systems is an attractive option as it employs existing tools with which animators are already familiar and which are part of popular modelling and animation packages.

The crowd system used by PDI/Dreamworks for feature films such as *Shrek 2* and *Madagascar* is an example of the use of particles [Ker05] (see figure 2.1). Their system represents each character in the crowd by a particle and uses a particle system tool to create the crowd's trajectories. Once this general motion has been approved by directors, specialised scripts, called High-Level Behaviours, are used to add additional animation to the characters. An example is the *look-at* script which retargets a character's gaze toward a particular point in space (some hero character for instance). Animation cycles are then applied to the crowd and adapted where needed for the High-Level Behaviours.

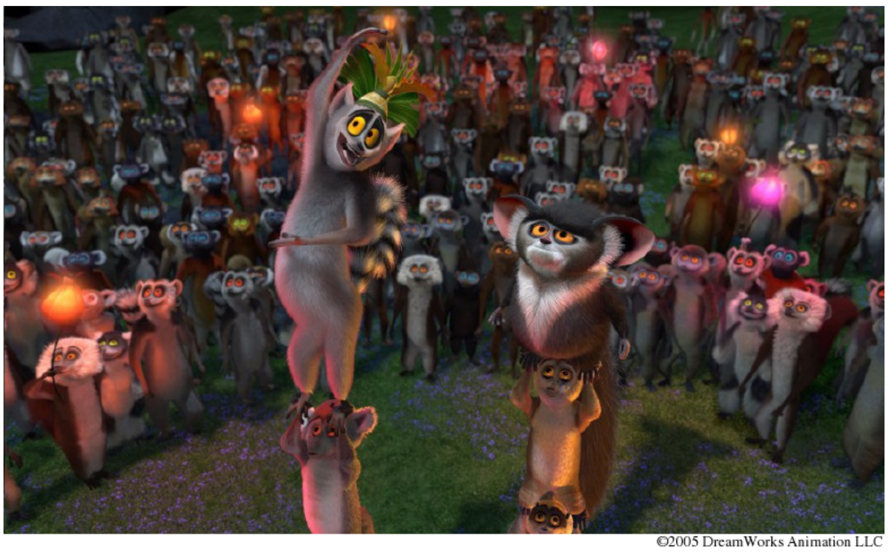


Figure 2.1: *A crowd from the movie Madagascar. The characters in the background were created using in-house crowd software by PDI/Dreamworks. Image source: [Ker05]*

A similar system was used at Industrial Light and Magic for films such as *Star Wars Episode 1:The Phantom Menace* [THL04]. Here again, particle systems with custom runtime rules are often used to animate the trajectories of the crowd members. Force fields in the environment provide a means to direct the crowd's path. In the description of the crowd animation process, it is interesting to note the emphasis given to the set up of the crowd. In some cases, custom tools were created to initialise a crowd for a particular scene.

Although these simple particle systems are used in film, they do not necessarily meet our described requirements. A simple particle system is quite flexible – the particles may be controlled quite intuitively using force fields and so the animator is free in creating different types of characters and behaviour. However, this simplicity reduces the *automation* of the system as much of the specification of a crowd's movement is left to the animator.

Another major disadvantage of this simple adaption of particle systems is the lack of behavioural complexity – it does not model the interactions between members of a crowd that we observe and expect. Crowds are made up of individuals that are aware of one another and so interact: pedestrians walking down a street avoid collisions and follow others that are moving in the same direction. The motion generated by standard particle systems lacks these characteristics. A crowd model that produces realistic crowd animation must take into account the characters of a crowd reacting to and avoiding one another, among other interactions.

Since the motion of particles in general particle systems is often controlled by force fields, one possibility is to have these controlling force fields take into account particle interaction. Chenny [Che04] introduces *flow tiles* which are combined into a divergence-free field. One possible application of this work is the simulation of a crowd by advecting particles through such a field. Since, by nature of the divergence-free property, the streamlines of the field do not cross, the particles will not collide. Although this deals with collisions between crowd members, the paths of the individuals cannot cross and so crowd interaction is severely limited.

A more accurate method is to directly model the interactions between individuals. In a physically inspired approach, Helbing and Molnár [HM95] introduce "social forces" which they apply specifically to pedestrians. These forces model the reactive, almost automatic movements that pedestrians make. When walking in a crowd along a street, the pedestrian is presented with well-known situations. People give little thought to their actions as they avoid collisions in such environments and so the authors argue that such actions are predictable and effectively captured by a simple force model. The forces used to describe pedestrian motion are: a force to accelerate to the desired velocity; a repulsion force away from other pedestrians and walls; an attractive force to keep crowds together. Using only these three forces, this social force model is able to reproduce observed crowd behaviour such as the striping that occurs when two streams of pedestrians intersect [HBJ05].

A more complex and restrictive crowd model based on physical simulation makes use of fluid dynamics. Hughes [Hug03] treats crowds as a flowing continuum, something similar to a fluid. This "thinking fluid" model represents a crowd as a continuous density field and describes the crowd dynamics with a set of partial differential equations. Hughes analyses the properties of the model but does not deal with the simulation of a crowd. Indeed, without any model of the individuals that make up a crowd, it is entirely unsuitable for a visual effects application where the members of the crowd must be individually represented.

More recently, Treuille *et al.* [TCP06] have used ideas from Hughes' work to create a crowd model based on continuum dynamics (see figure 2.2). The crowd is broken into groups, described as sets of particles moving according to a dynamic potential field. The potential field covers the environment that the crowd moves in and draws particles towards areas of low potential while repelling them from those of high potential. High potential is associated with obstacles and other particles and the field is updated every time step to allow for movement. The motion of particles is treated as an energy minimisation problem with a particle in the crowd moving opposite to the gradient of its global potential function. Although the calculation of the potential field is a costly process, it is shared by a whole group and so this technique is able to produce crowds of thousands in real-time. As the potential function takes into account other particles, the crowd members avoid collisions and move towards goals effectively.

Tantisiriwat *et al.* [TSK07] also make use of potential fields but combine them with Smoothed Particle Hydrodynamics (SPH), a technique often used for fluid simulation [MCG03]. This system decouples the environmental interaction from the crowd member interaction. A global path planning step makes use of potential fields and limited environment knowledge in order to determine a particle's path through the environment. Where Treuille *et al.* made use of a potential field for a whole group of particles, Tantisiriwat *et al.* use a simpler potential function which is calculated for each particle. The incomplete knowledge that a human has of its surroundings is modelled by calculating the potential field for each particle based on limited environmental information. This step further reduces

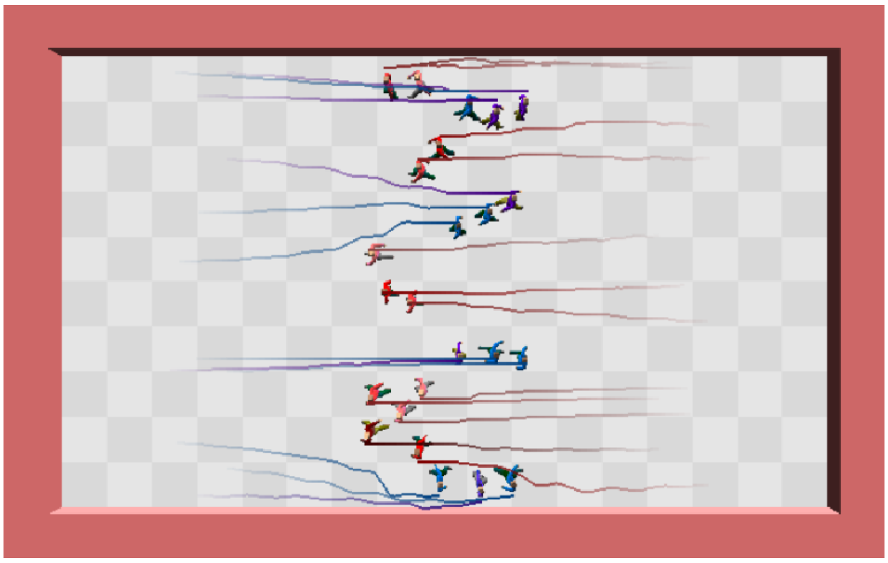


Figure 2.2: *The continuum-based crowd of Treuille et al. Image source: [TCP06]*

the expense of calculating the potential function and, since the potential field is based only on the static environment, the function need only be calculated when a particular particle's map of the environment changes. Particles gain further information by exploration and communication with other particles. Once a path has been determined by means of this potential field, the actual motion of the particle is determined by the SPH model which accounts for particle interactions.

An example of a particle model which takes into account crowd member interactions in a different way is described by Kirchner and Schadschneider [KS02]. These researchers move away from a force model applied to a spatially continuous world and instead simulate crowds in a discrete 2D environment using cellular automata, applying this model to pedestrians involved in evacuations. The relevant environment is partitioned into a regular grid with each cell containing either one or zero pedestrian particles. At each time step, each particle is able to move one cell along either axis of the grid, provided the target cell is unoccupied. The particle's movement is determined by a pair of transition probability fields that cover the floor of the environment.

The first of these fields is static and is used to guide the particles through the environment. In a room with a single door, for example, the probability field will slope towards the exit. This field alone will cause particles to move to cells that are closer to the door. The second of these fields models the interaction between the evacuating pedestrians. Since a particle's movement is constrained by the presence of other particles, collision detection is not needed. Instead, the interaction modelled is the tendency of pedestrians in an evacuation situation to follow one another. Each time a particle enters a cell, the transition probability of that cell is increased with the transition probabilities of all cells slowly decaying over time. The result is that particles tend to follow the path of those that went before them.

Although this model is interesting in that it reproduces behaviour observed in real evacuation situations, its usefulness in visual effects is limited. The discrete grid structure of the simulation is not realistic and so the quality of the animation produced is not acceptable for film. In addition, the interaction between particles is very specific to this particular area of application and the use of transition probability fields may be too simplistic to model interesting crowd behaviour.

2.2.1.1 Discussion

The particle systems described above take into account interactions between the members of the crowd to some extent. However, they have a number of undesirable qualities.

Since a particle is an entity with little more than position and velocity, a description of its motion is likewise simple. This is in contrast to the motion complexity of a character such as a human. The animation of a virtual character requires more than a description of its trajectory and so more information than is supplied by the particle-based crowd systems is required. The agent's trajectory could be used as input for some sort of constrained motion solver based on inverse kinematics or motion clips. Even so, it would be difficult for such a system to produce much more than motion related to the crowd member's locomotion and so it is preferable that the crowd simulation system create a full description of the character's motion. Because of its simplicity in crowd member representation, these particle-based methods do not produce the required *motion quality*.

The *control* of these physically inspired crowds is also somewhat problematic. As many of the particle crowd models are physically based, it is conceivable that physical constraint techniques could be applied to these simulations in order to provide control. This option has not been explored, however, and so control of a crowd's motion relies on the manipulation of the model directly. Since these models are based on physical systems, this is no trivial task and the control of these physically-inspired crowds suffers from the same problems as authoring such a crowd does.

Crowd animation needs to be created by non-technical persons and so the use of partial differential equations, potential functions and transition probabilities makes *authoring* difficult. Manipulating the parameters of such a crowd system is not intuitive and some understanding of the underlying system is required. Additionally, these descriptions of crowd motion are based on specific models that make assumptions about how the crowd members move and interact. This limits the *behavioural complexity* of the simulated crowd.

All of these particle crowd systems have the common advantage that they are *computationally efficient*. This is mainly due to the simplicity of the model and is less true for systems such as that by Treuille *et al.* where the physical equations that guide the system are more costly to calculate. As noted above, however, this simplicity is also a limiting factor.

The techniques inspired by particle systems are characterised by crowd members that do not move themselves. The crowd of particles are moved by some global force that may take into account the interactions between the members. This global controlling force is quite different from how crowd behaviour occurs in reality. A better imitation of real crowds is achieved by giving the crowd members the ability to act for themselves.

2.2.2 Crowds of Autonomous Agents

Real crowds are made up of characters who are able drive themselves in some way. The resulting behaviour of the crowd as a whole is determined by the actions of the individuals that make up the crowd. Where the particles of adapted particle systems are inert and unthinking, intentional action on the part of the crowd members converts them into *autonomous agents*. This thinking is what lies behind the original work on bird flocking by Reynolds [Rey87], shown in figure 2.3, which inspired much of the subsequent crowd simulation research.

Reynolds extends a simple particle system by adding orientation to the particles and the ability of the particles to exert forces in order to steer themselves. By observing the flight of a flock of birds, three simple forces are identified that keep the birds flocking together while avoiding collisions. The first force – *centring* – describes the tendency of birds to keep together. Countering this force is the tendency of birds to avoid coming too close to one another or any obstacles. Each bird exerts an *avoidance* force to steer itself away from any bird or environmental obstacle. The third force – *velocity matching* – also acts to avoid collisions and

Figure 2.3: *The flocking birds created by Reynolds. Image source: [Rey87]*

describes the way birds in a flock tend to match each other's movement.

Each of these forces represent an aspect of the bird's flocking behaviour. They may often be in competition, as is the case when a flock must break apart to avoid an obstacle. As the flock approaches the obstacle, the birds on a collision path with the obstacle will have their avoidance force accelerating them to one side or another while their cohesive force attempts to keep them close to their flock-mates. If a simple weighted summation scheme is used this may result in the forces cancelling each other out and the birds simply flying directly into the obstacle. The force arbitration scheme that Reynolds suggests prioritises the different steering forces and so orders their contribution to the resultant steering force exerted. In this way a more pressing need, avoiding an nearby obstacle for example, may override the other steering forces in an emergency.

A substantial extention to the flocking produced by Reynolds is made in the HiDAC system of Pelechano *et al.* [PAB07]. Their system, aimed mostly at high density crowds for evacuation simulation, implements a force model which takes into account not only collision avoidance and goal seeking, but also collision detection and response.

Goals, such as a doorway or exit point, serve as attractors which cause the agents to apply a steady force towards them. Obstacles and agents ahead of an agent are used to determine a collision avoidance force which is tangential to the object to be avoided. This force is weighted according to the object's distance from the agent. If the object is another agent then an additional weight is used which depends on the other agent's direction of movement. By weighting the avoidance force from agents which are moving oppositely higher than those that are moving with the agent, agents tend to move with the flow of the crowd. Lane forming emerges under high crowd density situations, similar to that observed with real crowds.

When an agent overlaps an object, that is, when an agent collides with something, a repulsive force that is perpendicular to the collision is produced. This enables agents to push other agents and the system even allows for the special case of an agent falling over when enough force is exerted. Further dynamic weighting of these forces allows the system to create variation among the agents and also induce different mental states, such as calm, aggressive or panic-stricken.

The simple agent-force model has also been combined with environment processing and path planning techniques in the system created by Pettré *et al.* [PKL08]. Their crowd simulation system, aimed at generating large crowds in real-time for virtual environments, makes use of a number of techniques in order to improve computational efficiency. Level of Detail (LOD) and Level of Simulation (LOS) schemes are used to reduce the work done on sections of the crowd based on their visibility. This approach is entirely unsuited to visual effects where reproducibility is key. If the granularity of the simulation is based on the position of the camera and the director changes the manner in which the camera moves, the behaviour of the crowd may change entirely. This creates a situation where the crowd would need to be tweaked for each camera move and so make authoring more complicated than necessary.

The environment is reduced to a graph structure made up of obstacle-free corridors which makes for easier long term path planning (further discussion of path planning in crowd simulation appears in section 2.2.5). The force-based behaviour is used only for inter-agent collision avoidance and to keep agents from moving out of their current corridor.

Unfortunately, these force-based techniques do not make for particularly easy authoring of crowds. This is especially true as the complexity of the model, with its associated weights and parameters, increases. The behaviour of the agents is expressed as weighted sums of forces, which an animator may find difficult to manipulate. Reynolds talks of the frustration of trying to get his birds to flock in the required manner as there is no method of control beyond directly altering the parameters of the crowd model. With the simpler models, there may be a lack of generality as is evidenced by Reynolds' birds which cannot be altered to produce, for example, pedestrian movement, unless additional forces are added.

In a change from the force model, Goldenstein *et al.* [GKM01] create agents which are able to control their orientation and forward velocity. The movement of each agent is modelled as a nonlinear dynamical system, with the orientation and velocity of each agent determined separately. The orientation function treats obstacles and other agents as repulsive forces and goal positions as attractors with their individual contributions combined by means of a weighted sum. The priority of the contributions is further determined by the shape of attractor/repellor functions, the repellor function having a steep falloff as it moves away from dead-ahead while the attractor is more gradual. The velocity of an agent is simply calculated based on the distance to the nearest object, with agents even able to reverse if they come too close to an obstacle.

The whole system is made scalable by the inclusion of a second layer which makes use of spatial data-structures to provides environmental information for each agent. An agent is supplied with information for only those objects that are close enough to be relevant. The result is that the complexity of an agent's movement calculations remains constant, regardless of the complexity of the entire simulation. In addition, the movement of an agent is calculated by simple function evaluations and is therefore computationally efficient. The drawback to this system is a lack of flexibility that makes authoring new crowds difficult. The authors themselves admit that the design of the attractor/repellor functions can be a complex task and so it is unlikely that non-technical users would be able to effectively tweak the behaviour of agents or create new ones.

Control of the crowd simulation is given by a third layer which provides global planning. This controls the goal positions which are in turn supplied to the local movement layer by the environment layer. By providing a general path for the agents to follow, a degree of high-level intelligence is created. These paths may be supplied by a user or some sort of global path planning system. For a visual effects application, one could imagine an animator supplying direction at this level. It should be noted, however, that this control is still subject to the local movement layer and so is limited to some degree. Control at a lower level is problematic as changing the local movement layer is a complex task.

All these systems, although making use of intelligent agents, suffer from some of the same problems as the particle approaches. The simulation deals only with the trajectory of an agent and does not simulate any other motion of the crowd members.

Blue Sky Studios, in the making of the movie *Robots*, used a simplified system based on the work by Goldenstein *et al.*. Their system [Pat06] omitted both the environment and planning layers, implementing only the local reactive behaviour. The high-level intelligence necessary for long term behaviour was not needed for short shots. In addition, the scalability added by the environmental layer seems not to have been worth the added implementation complexity.

The lack of full body motion was dealt with by assigning, either randomly or manually, a series of looping motion clips to each member of a crowd. The agent then followed its simulated path while executing the relevant motion clip. Control was added in the form of handles on each agent that could be manipulated by the animator. Additional handles were used to adjust the weighting of the contributions from the different attractors and repellors. Patterson notes that invisible obstacles were also used to control the path of agents.

Although the system for *Robots* proved successful, some of the automation of the system was removed in order to deal with shortcomings in the animation quality and control. This particular movie had only one type of crowd, however, the authoring of different crowds may have proved difficult due to the complexity of creating new movement functions as described above.

The crowd models discussed in this section produce individual motion which is combined through the interaction of individuals to produce crowd motion. This is important in that it allows a crowd to be created through the modelling of the individuals – a far simpler task than trying to describe the complexities of a crowd's behaviour as a whole.

As with the particle methods of the previous section, these methods suffer from an over-simplified representation of a crowd – the members of the crowd have only position and orientation to be animated. As seen in the production of the movie *Robots*, a large degree of automation is lost when the animator needs to do further work in order to create the full-body, high quality motion required by film. The behaviour of the crowd members is also limited by their representation as only position and orientation may be changed. The methods described make use of simple functions to describe this motion which are difficult to change and lack generality.

2.2.3 Crowds of Intelligent Agents

A characteristic of the members of most crowds is that they are able to react in an intelligent manner. We have thus far considered crowds made up of agents which are moved by an external agency and others that are able to exert some form of movement force themselves. We now consider research that adds an additional level of realism by allowing crowd members to react based on some form of direct, deliberative algorithm.

The social forces model of Helbing and Molnár, described in section 2.2.1, produces motion that should be collision free. Metoyer and Hodgins [MH04] modify this to create a pedestrian simulation system that allows the agents to follow a user defined path through the environment while still avoiding obstacles and other pedestrians. They notice, however, that the behaviour of the agents in the crowd is not always as expected. Pedestrians, especially in areas where the crowd density is low, generally avoid collisions by making small changes in direction well before a collision can occur. Although the social forces model ensures that the agents turn away from one another before a collision, the authors find that this movement is sudden and occurs later than one would expect.

Their solution is to include an additional behavioural model which may be trained by the user in order to provide more realistic collision avoidance. Based on research in the field of traffic planning, a number of behavioural primitives are identified. These are simple algorithms that pedestrians may execute to avoid a collision. Examples are *yield*: turning to allow another pedestrian to pass in front, *cut-in-front*: turning to pass in front of another pedestrian, and *go-around*: turning to avoid an area of possible collision.

Initially, the user decides which algorithm should be executed, with the system storing a situation vector and the user's decision. Using either a Bayes classifier or a decision tree, the system is trained to eventually allow agents to make these decisions for themselves. Each behavioural primitive produces a force which is added to the force produced by the social forces model in order to produce the agent's final steering force.

This approach produces animation that is 2D in nature - the agent's trajectory through the environment. A post-process is used to covert this to 3D motion by using a motion graph algorithm which selects motion clips that move the crowd member along its generated path. This is, however, limiting in that the motion is related only to locomotion. Although user-training allows easy authoring, the behavioural primitives are hard-coded and related only to collision avoidance. This lack of flexibility means that this crowd system is not able to produce as wide a range of behaviour as may be required.

A similar learning-based approach to crowd behaviour is described by Lee *et al.* [LCH07]. Their system makes use of live video clips of pedestrians in order to train an agent behavioural model. Agent behaviour is divided into two layers: high level and low level behaviour. A number of low level action models are arranged into a finite state machine with transitions between states being controlled by the stochastic process of the high-level behaviour. Each person in the video input is manually annotated to describe their high level behavioural state and then the relevant low level model trained with the particular person's actions in different situations.

An interesting question to be dealt with is which video features to make use of in determining an agent's action. The environmental obstacles are manually annotated and used along with information on the arrangement of other pedestrians as well as the individual's own speed and direction. This state vector is associated with the action that the individual takes in the given situation and is used as a training example for the relevant low level action model. At simulation time, a weighted linear regression model is used to combine nearby state-action pairs to provide the agent with the relevant behaviour based on its current state. Due to the nature of the information available from video and the need to keep the state vector of low dimension, the simulation can only produce trajectory information for a particular agent. This trajectory is then used as input constraints to a motion graph system (similar to that described above).

The problem with this technique is the need to produce a training video as well as the inflexibility of the high-level behaviour states – these are hard-coded into the system. In visual effects, however, this high-level layer may not be necessary as agents do not need to change high-level goals during a short shot. Their processing of the video input limits this technique to 2D input and output, though this may be somewhat dealt with by adding 3D motion to the trajectory information. The use of a particular state description vector may also prove problematic if different types of crowds are required.

Work by Lamarche and Donikian [LD04] builds on the previously described force model, attempting to produce more human-like collision avoidance behaviour in pedestrians. Their system includes environmental processing and path finding (see section 2.2.5). Using this, agents are able to plan a path through their environment to a specified goal. The agents navigate this path based on a force model which attempts to maintain an area of personal space around each agent by changing their velocity. On its own this is fairly similar to the social forces model of Helbing and Molnár (see section 2.2.1). However, in order to model agents which predict collisions and avoid them, a specialised collision detection module is added. Future collisions are determined, assuming that all agents continue at a constant velocity, and the nature of each collision categorised. An example of such a category is a *back-collision*, where the agent walks into the back of another pedestrian. Different strategies, such as slowing down or turning to the left or right, are used to attempt to avoid a future collision. Each suggested new velocity is scored according to the amount of change it requires and the highest scoring suggestion is used. Agents may be customised by specifying which strategies are used to suggest these velocity changes.

A more sophisticated version of predictive collision avoidance is given in later work by Paris *et al.* [PPD07]. Each agent predicts the position of any neighbouring entities for a number of discrete time steps of increasing length. For each entity, the orientation and speed ranges that avoid collisions are recorded in "orientation sections". In order to determine the eventual orientation and speed of a particular agent, different orientation sections are compared based on criteria such as how much orientation or speed change is required and how far these resulting state variables are from the agent's ideal.

An interesting property of this work is the effort to validate the model using real data. Motion capture is used to record the interactions between pedestrians whose paths cross, as well as for scenarios such as pedestrians entering a funnel shaped passage. The motion of these real pedestrians is compared to prediction from the crowd model. The results show that the model may be tuned to a particular situation, though it lacks the adaptation found in real pedestrians. Real people tend to alter their predictive behaviour based on crowd density, something that this model is not currently able to do.

Loscos *et al.* [LMM03] discard the force model entirely and instead take the important step of simulating the perception of an agent. Their work simulates pedestrians in a virtual city. The navigable parts of the environment are divided into a regular 2D grid which is used to efficiently store spatial information for each agent. An agent moves towards goals, defined in a separate path planning step (see section 2.2.5), while avoiding collisions by applying local rules. Each agent checks for obstacles, either part of the environment or another agent, in the area immediately ahead of it. If an obstacle is identified the nature of the possible collision is categorised (front, back or side) and a change in the agent's direction or speed is made according to the agent's navigational laws. These laws are in the form of *if-then* statements that take into consideration collision distance, the behaviour of other agents, the type of collision and the traffic congestion.

In order to produce more realistic pedestrian movement, the system models the tendency of pedestrians to follow the path of those ahead of them in a manner similar to that of Kirchner and Schadschneider (see section 2.2.1). For each grid cell, the direction of the last agent to leave the cell is recorded and used to influence the next agent that enters the cell. The strength of this influence decays over time and this effect only applies in areas of high crowd density.

The system also implements simple grouping by creating small group entities made up of a leader and a number of followers. The leader moves towards the goal as normal while the followers instead have the leader as their goal. Bunching is prevented by making use of the underlying grid structure to ensure that agents do not move into grid cells that are already occupied by other agents.

These five systems, all aimed at real-time pedestrian simulation, succeed in improving the realism of the crowd's movement while remaining computationally efficient enough to simulate thousands of pedestrians in real time. They add behavioural realism in the form of specific collision avoidance algorithms that mimic human reactions. The disadvantage of such a feature is the inflexibility of these algorithms. It works well for pedestrian simulation, but may not generalise to other types of crowds and would be difficult for a non-programmer to alter. Although behavioural complexity is improved, ease of authoring suffers as the system is now less flexible.

In addition, these systems retain many of the problems of the force-based particle approaches. Most importantly, the animation they produce consists of only orientation and position information though this may be converted to full bodied motion to some degree using a post-process. Related to this is the simplicity of the behaviour which consists almost entirely of goal seeking and collision avoidance. As the aim of these system is the simulation of crowds without requiring supervision, control is not emphasised and requires the manually altering an agent's behaviour. This altering of the behavioural decisions is not guaranteed to have an intuitive link to crowd behaviour and requires programming skill. The agent's actions are simple and so individual controls could be added to override the simulated behaviour, providing low level control.

The system described above add intelligence in the form of specific behavioural algorithms. A better approach is the generic and extensible framework for agent-based crowds is proposed by Niederberger and Gross [NG03]. By abstracting the different aspects of an agent's behavioural process, the authors are able to create a system that is general enough to be used for a wide range of character types. The behaviour of an agent is based on situation recognition and models an agent's perception, deliberation and actions, as separate entities. At a particular time-step, an agent receives information about the world through a sensor interface. This information, in the form of boolean statements, along with an agent's internal state variables, is then used to determine the situation that the agent faces. Since this information may match more than one situation, each situation has a set occurance probability and the most likely situation is used. An agent's behaviour is defined by the situations that it knows how to deal with. Each situation has a number of actions associated with it and the most appropriate one is selected based on the perceptual and internal information.

The sensors, situations, internal state and actions can all be customised to some extent by setting attribute values, allowing agents to vary slightly. The crowd framework is hierarchical and allows multiple inheritance so that new agents may be created by combining existing ones. Agents with a few simple behaviours may be combined and varied to create a crowd of agents with more complex behaviour. This property makes the authoring of crowds significantly more efficient. Another important improvement from an authoring perspective is the XML-based specification of an agent's behaviour, consisting of situations and associated actions. Although the sensors and actions must be hard-coded, an agent's behaviour may be designed by a user without programming knowledge, especially if some form of user interface is created to edit the XML files.

This general situation-based framework allows for greater flexibility in behaviour as the behaviour of an agent is now customisable. Unfortunately, the behavioural complexity is limited somewhat by the fact that only one situation is activated at a time and only one action undertaken. Although the authors suggest that the system could be extended to include multiple inheritance for situations and actions, this has not been explored. The control of the crowd simulation is only at a behavioural level - the thinking of the agents must be altered directly. This may be done by altering an agent's attributes but this has not been studied and may suffer from an unintuitive link between attributes and crowd behaviour.

The system implemented by the authors is only tested on a few hundred agents and so it is difficult to say how much of an impact the more general framework has on the computational efficiency of this crowd model. The computational cost of an agent is tied to its sensors which give it information about the surrounding world. The time complexity of these sensing routines is, thereofore, likely to be dependent on the complexity of the environment.

An interesting possibility is created by the abstraction of an agent's actions. Although the implemented system only seems to have actions which change the agent's orientation or position, the crowd framework described is not as limited. Where as the other crowd models discussed have been able only to determine an agent's trajectory, usually by a force applied to the agent as a whole, the situation-action relationship does not impose such a limitation. This allows us to consider what other actions an agent might take and considerably expands the behavioural complexity as well as animation quality.

2.2.4 Crowds of Embodied Agents

The crowd models discussed so far all suffer from a similar problem related to the representation of the crowd members in the simulation environment. In almost all cases each crowd member, however complex its behavioural model, only exists as a point in virtual space. As discussed previously, the animation produced by the crowd simulation is tied to this representation and so the motion is a simple trajectory to which more complex and interesting full-character motion must be added. An important step in producing high quality crowd simulation such as that needed for film is to give agents a body. The behavioural model of an agent is its brain, deciding what it should do, but a brain without a body cannot produce interesting motion. More believable, realistic crowds are produced by embodied agents.

The use of embodied agents raises a number of issues that have been addressed in different ways. Perhaps the most important of these is the brain-body interface – how does the behavioural model of the agent tell the geometry of its body what to do? Having a single system which queries the world for information, decides the actions that an agent should take and then fully describes the relevant joint rotations or torques is impractical as this would be too complex for an animator to use. The usual approach, adopted to different extents and in slightly different ways, is to separate the perception, decision and action components of an agent's behaviour.

An example of a crowd model that uses very limited communication between the decision and action components is the approach developed by Brogan and Hodgins [BH97]. Their work investigates how the dynamics of an agent's body affects the performance of a behavioural model. They look at crowds of agents with dynamically simulated bodies – one legged robots and cyclists shown in figure 2.4 – and compare these results to those produced by a simple point-representation similar to that of systems in previous sections. Because the agents must now take into account constraints such as balance, their reaction time and ability to react impact the behaviour of the crowd.

The behavioural model is simple and produces flocking similar to that of

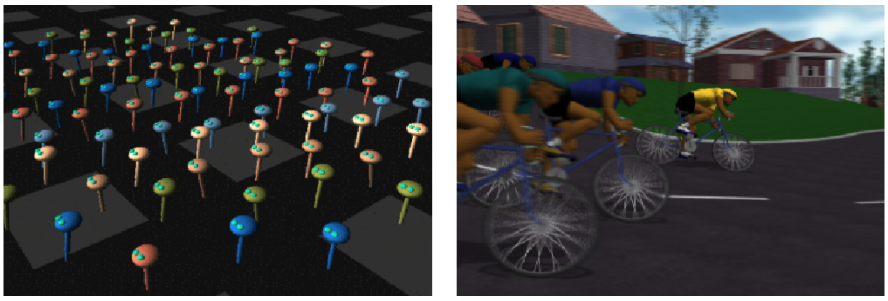


Figure 2.4: *Embodied one-legged robot and cyclist agents. Image source: [BH97]*

Reynolds (see section 2.2.2). The behavioural model consists of an perception and decision algorithm that determines a desired future position for the agent, based on the positions of its neighbours, its velocity and any nearby obstacles. This perception and decision step is identical for all agents, regardless of their type.

The action step, however, is dependent on the model of the agent's body and so differs substantially from one type to another. The point-mass agent simply applies a force calculated to accelerate the agent towards its desired position. The robot and cyclist agents each have physically-based models and control algorithms that attempt to alter the agents velocity such that it reaches its desired position. It is important to note that it is very likely that an agent is not able to reach the desired position because of the physical constraints of the simulation. In this case the agent does its best - the definition of which is determined by the body control algorithm.

The use of embodied agents, even with this simple behaviour, produces significantly more interesting animation than would otherwise occur. The robot agents hop and the cyclists move their legs and bodies though the simulated nature of their body dynamics means that the motion produced may appear somewhat robotic. Although the perception and decision aspects of the agent's behaviour is not computationally costly, the simulated dynamics and related control means that the action step introduces significant computational overhead. In addition, the incorporation of this action step, which is not discussed in detail, requires a considerable amount of implementation work for each type of agent.

The perception and decision parts of the above behavioural model are simple and specific, which means that they are inflexible and only aimed at a single behaviour. However, their use for different agent types illustrates the animation quality that can be gained with little intelligence on the part of the agents. The control of these agent's behaviour is not provided for except through manually altering the behavioural algorithm.

An impressive and detailed model of fish with more complex behaviour and perception is presented by Tu and Terzopoulos [TT94a]. Using an artificial life approach, this behavioural animation model makes use of perception, decision and action systems that are modelled on real fish. The perception system, in particular [TT94b], allows a fish knowledge of other fish and obstacles in a 300° forward arc to a distance based on the murkiness of the water. The fish, modelled as spring-mass systems, are propelled by the contraction of muscle springs. This low-level action is abstracted by motor controllers which are available to the decision system of the fish agent. This system determines intentions or goals for the fish and then attempts to move the fish toward completing their current goal.

The fish may be individually characterised by setting parameters that are related to a fish's tendency to choose a particular goal. Examples of these goals are *mate*, *escape danger*, *feed* and *wander*. The intention generator is hierarchical in nature so that more pressing goals, such as collision avoidance, are selected before other less pressing ones, such as feeding. Once an intention is determined, the relevant behavioural algorithm is called, which in turn activates the parameterised motor controllers of the action system. Different fish types are created by giving the fish agents different intentions or by varying the behavioural routines. For example, predator fish do not have any escape intention and would look for another fish when executing the *feed* behaviour, while prey type fish look for some form of plant food.

The crowd motion that is produced by this system is interesting in that the group behaviour becomes more complex than that of the individual. This is a form of emergence, which brings with it advantages as well as complications. The quality and realism of the fish animation is enhanced and the fish seem to have a life of their own. This does make the simulation difficult to control as there is no longer a direct and clear link between the specified behaviour of the fish agents and the actual crowd behaviour.

Although the system is able to produce complex interactions, it suffers from inflexibility and is limited to the simulation of fish. The behaviour routines, perception and action systems are all hard-coded and therefore impossible for a non-technical user to change. The fish behaviour may be customised somewhat but most variation is achieved through selecting behaviours from a relatively small, fixed set. The action system is effective but benefits from the relative simplicity of a fish's body and, even so, introduces a considerable performance penalty. More complex characters, such as humans, would make this physical simulation approach difficult.

Later work by Shao and Terzopoulos [ST05] is forced to address this issue in the simulation of pedestrians in real time. Their approach is to replace the physically simulated body of the fish with a body animation sub-system that makes use of linked motion clips. Although the authors state that the motion achieved by this sub-system is not of the highest quality, actions are requested via a general interface that allows an improved version to be added without difficulty. It is likely that further motion, collected using motion capture, and motion blending techniques such as motion graphs [KGP02] could improve the animation quality. The use of a data-driven motion clip approach in place of a physically-based simulation reduces the computational cost of the action system while improving the quality of the animation produced.

The perception and decision systems apply similar principles to the fish modelling described above. Perception is broken into two parts - the separate sensing of static moving objects. Static sensing is achieved using a ray-casting approach which gives the agent information about objects that lie along line segments shot out in a fan from the agent's position. This query is kept efficient by making use of a fine grid-based environmental structure. A coarser grid is used for keeping track of moving objects and each agent queries this grid directly, selecting cells in a fan shape immediately ahead of itself.

The decision system is made up of a number of basic behaviour routines on which more complicated behaviour is built. Agents make use of high-level path planning and long term goals to move through the environment, while relying on the lower-level behaviours to avoid collisions. The basic behaviour routines make use of perceptual information and then issue motor commands to the action system. These routines are executed in series, with higher-priority behaviour occurring later so that each behaviour may alter the motor commands given by the behaviours before it. In this way more pressing reactions override the less immediate ones.

A sophisticated navigation and path planning module is built on top of the basic behaviours. This system builds on previous work in cognitive modelling and autonomous navigation to allow the agents to move through their environment, attain goals and determine new ones. There are also specific higher-level behavioural routines that produce realistic behaviour in crowded passages and other non-navigational behaviour such as making a purchase from a vending machine or chatting with a friend. These behaviours, even more so than the low-level behavioural routines, are specific to pedestrians and, in some cases, even specific to particular environments. This means that the system would have difficulty in producing varied crowds, since adding new behaviour requires programming new behavioural routines.

The pedestrian system described above meets a number of the requirements for film. There is the potential for high quality animation and realistically complex behaviour. The computational efficiency is sufficient to animate thousands of pedestrians in real-time and the agents, once created and initialised, are animated completely automatically. There are two main areas where the system fails, both more important to the system's use in film than for other applications. These are *ease of authoring* and *control*. Both are related to the difficulty in changing the crowd behaviour.

2.2.4.1 Improved Control

The crowd models that have been discussed suffer almost universally from a lack of control. This is due in part to the fact that many of these models are aimed at applications that differ in significant ways from film – virtual environments and games, for example. As previously stated, control is of great importance to a crowd simulation's usefulness, since a crowd that cannot be controlled may not behave as required and so fail to contribute effectively to the telling of a story.

One approach, first developed by Funge *et al.* [FTT99], is to give the crowd members greater intelligence. They introduce a cognitive layer on top of an agent's behaviour, which serves as an additional level of abstraction on top of the actual animation of the character's geometry. The cognitive layer of an agent allows the user to specify goals or constraints from which the agent will work out a detailed sequence of actions to meet the user's specifications.

Although this approach is reasonable and effective for individual goals and constraints, the action planning takes place on an individual level. This means that the control relies on an individual's ability to achieve the specified goal. Even if the same goal is specified for a number of agents, there is no guarantee that agents will work together to achieve it. This approach falls victim to the adage that, while an individual may be intelligent, a mob is stupid. There is also a large degree of complexity added by this layer. The planning steps for a large crowd may be computationally unmanageable and there is also an additional price to pay in authoring new crowd types. Each new agent must now be told not only how to react, but also how to plan.

Blumberg and Galyean [BG95] also research the control of behaviourally animated characters. Their aim is to provide a degree of directability on multiple levels of an autonomous character's behaviour. Their behavioural model, designed more for single characters than whole crowds, includes such high level aspects as motivation. Control is introduced at the levels of motivation, decision or action, by directly altering control parameters at these levels. This control is useful for individually tweaking and directing the behaviour of agents. It may be used to turn the one agent who keeps falling off a cliff, for example, or change the actions of a particular agent that is too noticeable. The introduced control, however, is not easily extendable to whole crowds, as changing the behaviour of a group of agents is not always easily linked to a change in the behaviour of the crowd as a whole.

The crowd-level control introduced by Chang and Li [CL08] is of interest both for their underlying crowd model and the control added. They model crowd members as fuzzy logic controllers making use of a number of fairly simple fuzzy rules to link senses to actions, in an approach based on that used by Tunstel *et al.* [TdB02] for the control of autonomous robots. Their crowd behaviour model seems similar to that implemented in this dissertation, although the details of the system are not provided. The behaviours built into their system are specifically aimed at shape control (see figure 2.5), which is the main contribution of their work.

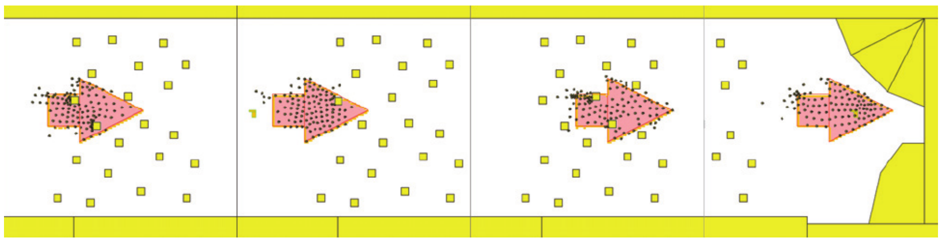


Figure 2.5: *The crowd shape control introduced by Chang and Li. Image source: [CL08]*

The goal in shape control is to be able to move a crowd through an environment, avoiding obstacles while maintaining the space formation specified. This is accomplished in two steps by first planning a path through the environment such that the centre of the shape avoids collisions. Once the shape's path is known, the deformation of the shape is determined such that it does not overlap the environment. This deformed shape is then used as an input to the crowd simulation, similar in control to a force field. The agents behaviour dictates that they attempt to remain inside the shape, resulting in a crowd of agents that roughly conform to the described shape. There are, however, some instances where the crowd coverage across the shape is less than optimal.

This example of high level control provides a specific tool for shaping crowds. The control introduced, though useful, is not general. The shape control is an example of bottom-up control in which the goal of the control is given to the agents and they are required to meet it. This is similar to the cognitive control of Funge *et al.* which can be somewhat unreliable. Each agent has the goal of filling the shape but there is no cooperation in this task. A situation may occur where an agent enters the shape and then, his goal satisfied, inadvertently blocks the way for other agents.

2.2.4.2 Improved Authoring

Crowd simulations must be flexible enough to be used for a wide variety of characters. Additionally, they must provide a means to tweak the crowds quickly and ideally without requiring much technical knowledge of the inner workings of the crowd simulation system. This is not the case for most of the current crowd systems developed, although some work has been done that attempts to make the authoring of crowds easier.

Improv, a system created by Perlin and Goldberg [PG96], is related to this aim. Focussed on interactive animations involving virtual actors, Improv has a number of components which are geared towards ease of authoring. Although not aimed at large numbers of agents, some of these authoring advances are relevant to crowd simulations.

The behaviour scripts that drive the animation of the actors are layered in a manner similar to that of Niederberger and Gross (see section 2.2.3). This allows lower-level behaviour to be reused, combined and customised into more complex behaviour. The other important addition is the manner in which different actors may communicate with one another using a blackboard. This blackboard allows one actor to tell another how to respond when they interact. This allows actors to cooperate and choreograph actions as directed by the author.

The blackboard used by the Improv system is similar to work done by Sung *et al.* [SGC04]. Whereas Perlin and Goldberg allow actors to tell one another how to respond, this system enables the environment to dictate how agents act. Behavioural routines or rules are embedded in the environment using a paint-on technique and agents who move into the designated area then have the particular behaviour added to their behavioural model. This approach is scalable in that agents are only as complex as they need to be – they are assigned behaviours as required. This is relevant to the authoring of crowds, particularly in virtual environments, in that crowds may be created by specifying the environment once and then allowing crowds to come into contact with it. However, this work is of more use to applications where crowds and environments are in existence for longer than the short shots used in film. The number of different situations that a crowd member may be in over the length of a shot is limited.

An interesting concept for the interactive authoring of crowds is presented by Ulicny *et al.* [UHT04]. Inspired by the immediate feedback loop in game and other design applications, a "crowd brush" is designed that allows a crowd to be directly painted onto a scene or environment. The user is able to specify the action that the brush takes. These range from crowd creation/deletion, through placement tools to behavioural specification. The brush is also able to randomly vary attributes of crowd members and can be customised to change any detail of the crowd. The brush itself works in 2D and has its effect mapped to 3D in real-time. A particular requirement of this technique is that feedback be available interactively. For crowds made up of characters with complex geometry and behaviour, some form of imposter scheme may be necessary to achieve this.

The most successful and widely-used crowd simulation system in visual effects at present is Massive. Developed to animate the huge armies of *The Lord of the Rings Trilogy* [ABL04], Massive has been widely used in over 50 full-length movies and many other short films and commercials[1]. The system is more fully described in chapter 3, but the authoring tools that are provided are worth noting. The agent brains are made up of fuzzy if-then rules which are represented as a graph. This graph is interactively edited by a user who is provided with the structure of the brain. The technical details of how the brain inference works are largely hidden from the user who only has to deal with brain nodes that may be linked together to form logical chains. The use of fuzzy logic is also important in the authorability of the crowd as this links well to the way that humans think as it deals with values that are specifically modelled on human language.

2.2.5 Top-down Approaches

The basic approach of the crowd animation research described in the previous sections is bottom-up in that it aims to simulate the individual in order to create the whole crowd. This strategy is understandable, especially for visual effects ap-

[1]see www.massivesoftware.com

plications, as the individual needs to be animated in the final product. There are, however, a number of top-down techniques that are relevant to crowd simulation, although not all of them are complete crowd animation systems in their own right.

Motion capture is widely used in visual effects because of its ability to capture the nuances of articulated motion. Capturing the motion of a complex system, such as a human body, is easier than simulating the system itself. The success of this data-driven approach has led to research in a similar direction for the animation of crowds. Group Motion Graphs [LCF05] are an extension of Motion Graphs [KGP02], a technology used to stitch together motion capture clips. Motion Graphs arrange clips of motion capture in a graph, with edges representing individual clips and nodes representing decision points for a decision between clips. Unbroken motion can then be generated by traversing the motion graph.

Group Motion Graphs (GMG) apply this idea to group motion by taking clips of crowd motion and clustering them based on the local arrangement of the individuals in the clip. These clusters are stitched together and arranged in a GMG in a similar manner to a Motion Graph. Group motion may then be created by traversing this graph. Although a GMG, once constructed, provides a method for quickly generating motion that can be constrained arbitrarily, there are a number of problems with this technique, particularly for visual effects.

An issue with the creation of the GMG is the source of the group motion. In the author's implementation this motion is taken from a crowd simulator that generates particle motion, similar to Reynolds' birds (see section 2.2.3). If the GMG is used with a crowd model that produces full bodied motion, there would be a complication in clustering the group motion clips. If only the group members position and orientation information are used, it is difficult to link individuals that are similarly arranged but take different actions. If the clips are clustered on the the configuration of their bodies, it is unlikely that different clips will be linked easily and the difficulty of creating transitions for full body motions exists.

Motion capture is highly successful because, although the motion of a human body is highly complex with many degrees of freedom, there are a relatively small number of motions that are taken and they are fairly easily grouped. This in not the case with crowd motion and the space of possible group motion clips increases dramatically. Group motion also lacks the "rest" poses that are useful for linking motion clips due to a lack of the parameter constraints that are present in the joints of the human body.

A top-down technique that has been referred to in previous sections but largely ignored so far is that of high-level path planning. A number of crowd simulation systems, particularly those designed for use in virtual environments, make use of path planning in order to provide the agents in the simulation with a path through the environment that avoids static obstacles. Examples of such systems include the topological structuring and path finding of Lamarche and Donikian [LD04], the quad-tree based global path planning of Shao and Terzopoulos's Autonomous Pedestrians [ST05] and the collision-free corridor graph searches of Pettré *et al.* [PHM06].

The use of such long term planning and goal seeking behaviour is not relevant for visual effects due to the length of the shots and the control required by directors. The high-level decisions are made by the animator rather than the agents themselves as the application requirements of the crowd are different. In virtual environments the crowds have to behave like a viewer or user would expect them to, over both the short and long term. In visual effects, the crowds are required to behave exactly as the director requires. In the short term it is useful to make use of behavioural animation, however, without any knowledge of the story being told, the crowds cannot be expected to know how they should behave in the long term.

A top-down approach to control described by Lau and Kuffner [LK06] is of special interest here. Their system is able to impose hard constraints on the motion of multiple characters. The implication and usefulness of this control is something that has not been widely explored in the field of crowd simulation. This system implements control by means of motion graphs similar to those of Kovar *et al.* [KGP02]. These graphs capture the possible movement, and hence future positions, of a character and can be searched in order to find paths through the graph that meet particular constraints. This searching of the motion-space of an individual is accelerated by means of a grid placed over the world covering all the possible future position of the agent.

The considerable advantage of this technique is the degree of control over the agents being animated. If the motion graphs of a group of agents allow them to reach a particular position then the user can force them to do so. The disadvantages are related to the manner in which the motion of a character is generated.

In the absence of constraints, or with constraints that are not difficult to meet, the agents have no behaviour of their own. The motion generated is entirely defined by the motion graph and there seems to be no behavioural model for determining the path through this graph. The result is a substantial loss of automation as constraints must be placed on all the agents at all times in order to get them to move or interact.

The second drawback is that control extends only to the position of agents in the virtual world. Because the acceleration structure makes use of world positions in order to index parts of the motion graph, the constraints must be specified in a manner which is compatible with this optimisation. Without such an optimisation, the search of the combined motion trees of all the agents in a crowd would not be manageable.

Despite these drawbacks, the power of this system in exactly specifying the positions of individuals is attractive. No other crowd simulation system allows this level of control. It is in this direction that our work lies.

2.3 Discussion

A crowd is made up of a number of individuals that are each able to sense, think, act and plan. Most crowd simulations model this reality with differing degrees of abstraction. The range of models, from simple particles that require little computation but consequently provide little automation or information, to the sensing, reacting embodied crowds created by the Massive system, are appropriate for different applications. A summary of our findings is shown in table 2.1. We observe that the more complete models of individuals provide crowd simulations that are more useful in the visual effects industry.

	Particles	Autonomous Agents	Intelligent Agents	Embodied Agents	Massive
Animation Quality	low	low	low	high	high
Control	high	low	low	low	medium
Automation	low	medium	medium	high	high
Ease of Authoring	high	medium	low	low	medium
Behavioural Complexity	low	medium	medium/high	high	high
Computational Efficiency	high	medium	medium	low	low

Table 2.1: *A comparison of crowd simulation categories based on the requirements of visual effects.*

The key requirement for crowds in film is that of animation quality. This is provided by giving the agents in a crowd a body which can be moved by the decision making process. By allowing the self-animating nature of the agent's behaviour to move the agent's body as well as simply change its location and orientation, the limits on the quality of animation produced by simpler systems are removed. The important step in giving an agent a body is providing a brain that can move the body. A general description of actions in a brain is therefore required as opposed to the more specific force-based models.

Almost all the crowd systems reviewed provide automation to the task of specifying a crowd's motion. The degree to which this automation is successful is largely determined by how well the crowd simulation is able to meet the application's requirements without manual editing. Simpler crowd models may be unable to achieve enough behavioural complexity, requiring that additional animation be added manually.

The use of Massive, one of the most complicated and general crowd systems described, is illustrative of the degree to which computational efficiency is of secondary importance. The off-line nature of visual effects means that more complicated simulations are allowable if the quality of the outcome is improved. Although the trade-off between quality and speed still exists, quality of both motion and behaviour is the preferred variable.

A final factor in judging the suitability of different crowd models for visual effects is the use of the model by the visual effects industry. Although the wide use of a system does not necessarily mean that it is the best method, there is little competition with Massive in the crowd animation market. The widespread use of virtual crowds in both movies and shorter productions is based largely on the usability and quality of the crowds created by Massive. Any work that seeks to advance visual effects crowds must take into account the success of this system.

The one requirement which has consistently not been met is that of *control.* Even Massive provides only limited control of agents using simple force fields and paths. The usability introduced by the graphical representation of the agent brains and the clear link between fuzzy logic rules and linguistic concepts allows a user to manually alter a crowd's brains. This approach is limited, however, as discussed in later sections.

CHAPTER 3

Massive

Massive, the crowd animation system originally created for the huge battle scenes needed in *The Lord of The Rings*, is something of a visual effects industry standard (see figure 3.1 for an example crowd). As discussed in the previous chapter, Massive meets many of the requirements of a visual effects crowd animation system: high motion quality, automation, ease of authoring and high behavioural complexity. This is verified by the clear acceptance of Massive in the visual effects community.

Since Massive is proprietary software owned and developed by Massive Ltd, details of the system are not publicly available and much of how it works can only be inferred. What follows is an attempt to reverse engineer the important principles behind Massive, based on what can be gleaned from literature and from the author's experience of working with the Massive platform.

3.1 Overview

Massive simulates crowd members as autonomous, intelligent, embodied agents [THL04]. Each agent reacts to the environment and other agents according to its own thought process and what it is able to sense. Like some of the more general and complex crowd simulation systems discussed in the previous chapter, a simulation step of a Massive agent consists of perception, decision and action sub-steps. This process is realised by two separate entities: the brain and the motion tree. Although Massive shares much of this information between agents for efficiency purposes, from a conceptual point of view, each agent may be considered as having it own brain and motion tree.

The animation of an agent is created from motion clips that are imported into Massive from a character animation package or motion capture data. This library of motion data is organised into a motion tree which determines which clips may be linked together [Dun02]. Motion blending is used to keep the motion fluid and

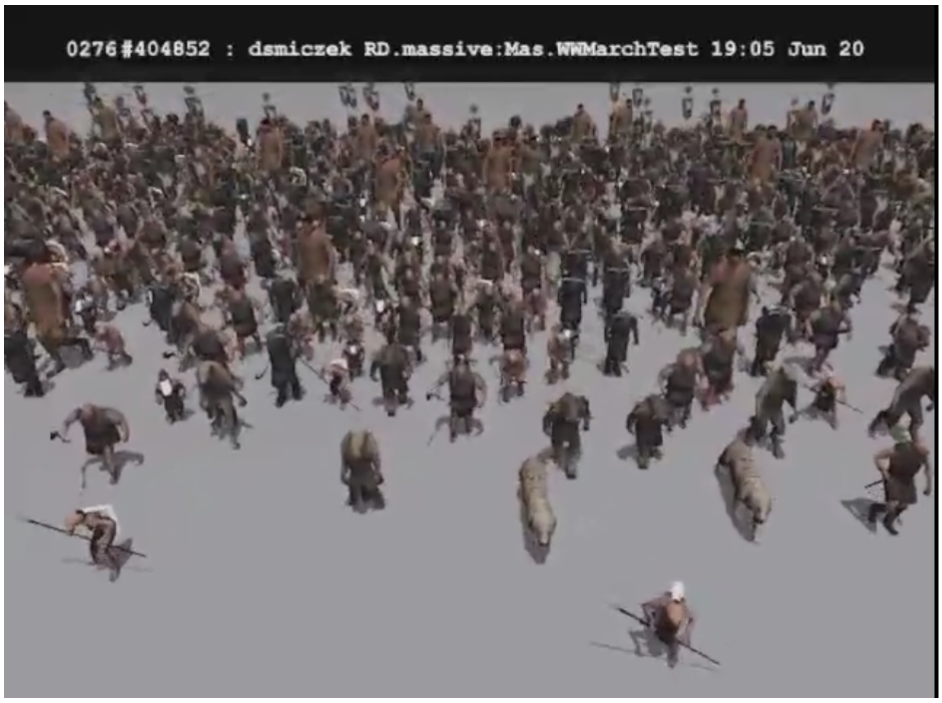

0276#404852 : dsmiczek RD.massive:Mas.WWMarchTest 19:05 Jun 20

Figure 3.1: *A crowd generated using Massive during the making of Narnia. Image source: http://www.massivesoftware.com*

even combine different motion clips. The choice of which actions to take is made by the agent's brain. The motion tree then applies this motion to the agent's geometry [Gri03].

The brain of an agent combines an agent's perception using fuzzy logic rules, and then informs the motion tree of the action that should be taken. The senses that make up an agent's perception model real world senses such as vision and hearing [Dun02]. The rules make use of fuzzy logic which mimics the subtlety of human responses which are not boolean in nature. The results of the fuzzy inference are combined, defuzzifed and sent to the motion tree, which then performs the appropriate action.

In creating a crowd of agents, the usual approach is to produce a brain for a particular type of creature. This "master" agent is then instanced, with possibly some variation, to produce a whole crowd of similar agents. The motion tree of the agent is also created and instanced in a similar way, with variation introduced across the various instances of a particular master agent [Rob08].

3.2 Senses

Creatures in real crowds have only limited information available to them. Factors such as visibility, distance and occlusion result in partial knowledge of the environment and other characters. Massive mimics the sensing abilities of real creatures in an attempt to produce more realistic and believable crowd behaviour. Two of the senses often described are based on vision and hearing.

Agents are given sight by providing each agent with a simple 2D rendering of the 3D scene from the agent's point of view. This synthetic vision approach is not new, different forms of the idea being described by Blumberg for a virtual dog character [Blu97] and by Noser *et al.* [NRT95] for virtual actors. Its use in crowd simulations is relatively novel, however, and an important step in that it provides an agent with a realistic amount of information while remaining scalable and efficient.

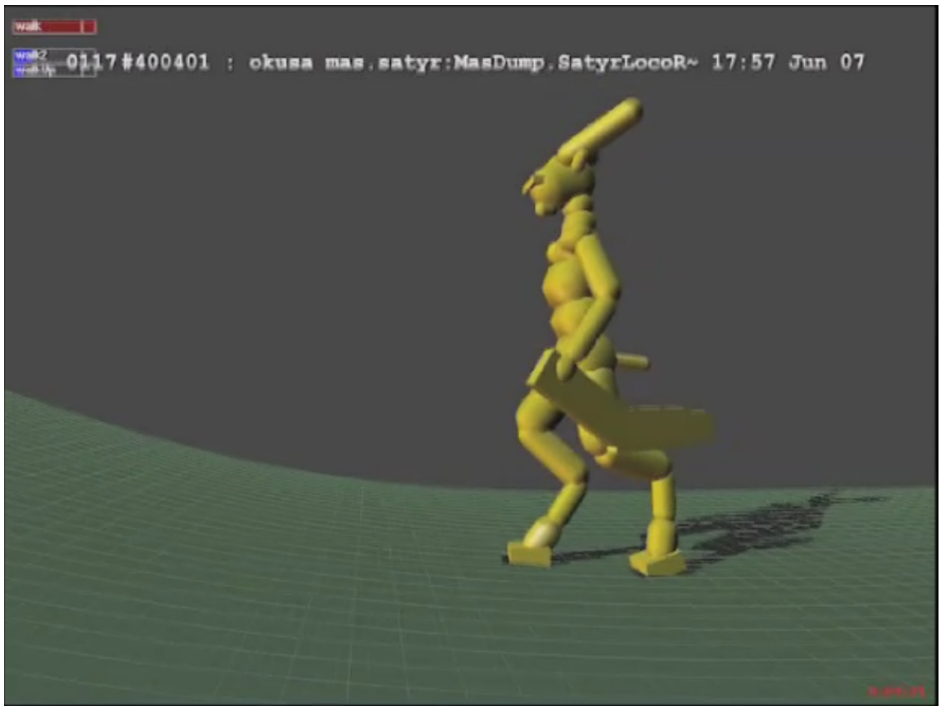

walk
0117 #400401 : okusa mas.satyr:MasDump.SatyrLocoR~ 17:57 Jun 07

Figure 3.2: *A Massive agent rendered using simplified geometry. Image source: http://www.massivesoftware.com*

Rendering is a topic widely studied in computer graphics and many optimisations have been developed aimed at keeping the computational cost fairly constant, even as the complexity of an environment grows. There is also specialised hardware which makes the rendering of a scene, especially one using simplified geometry, false colour and no lighting effects, computationally very efficient (see figure 3.2).

The sense of hearing is simulated more simply. Each agent emits a sound that is unique to its type. Nearby agents can hear the sound, know that there is an agent in that direction and also what type of agent it is. This hearing sense was found to often be sufficient for navigation purposes and is even less computationally costly than vision [Dun02].

3.3 Brain

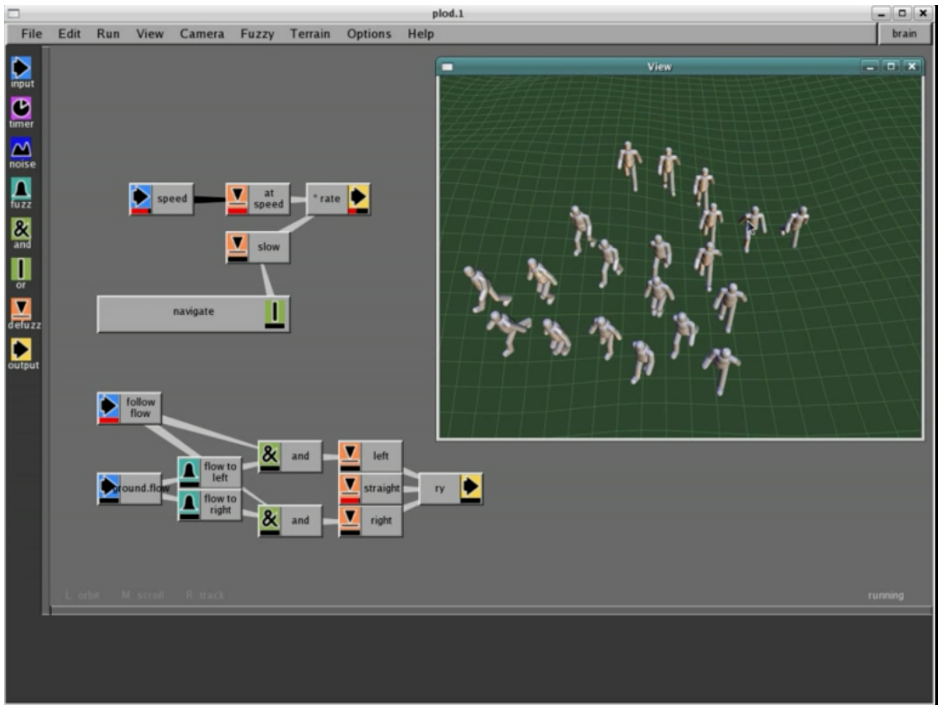


Figure 3.3: *A simple brain network in Massive with nodes connected as a directed graph. The "navigate" node is a placeholder for a collection of nodes defining navigation rules. An example crowd is shown inset. Image source: http://www.massivesoftware.com*

The brain of an agent is represented by a directed graph of different brain nodes – a simple Massive brain is shown in figure 3.3. These nodes may be roughly divided into four types: input, logic, defuzz and output nodes. The inference mechanism of an agent's brain makes use of fuzzy logic, a form of logical inference which uses continuous values as opposed to the two, on or off, values of boolean logic (see chapter 4). The input nodes convert information from one of the agent's senses into a fuzzy value that may be used by the brain. This is done using continuous membership curves that map the perceptual input to a fuzzy value. These inputs nodes are then connected and combined using logical nodes, which perform fuzzy inference operations analogous to those found in standard boolean logic such as AND, OR, NOT. The inference process may result in more than one value for a particular control variable and so defuzz nodes are used to combine the multiple values into a single one. The results of this inference process is then fed into an output node which sends the value of the control value to the motion tree in the form of a weighted trigger [Gri03].

A typical brain for a Massive agent in the *Lord of the Rings* movies contained 7000 to 8000 nodes [Dun02]. Although the brain nodes, in fact, make up fuzzy if-then style rules, the graphical user interface (GUI) is important in that it allows animators and non-programmers to edit the brain of an agent. Organising the 2000 to 3000 rules that these nodes represent, without the use of a GUI, would be a challenging task, even for a user familiar with the working of the system.

The nodes themselves represent fuzzy variables which have differing degrees of truth. A particular input node might be called "wall-left" and take on a value based on a particular membership curve applied to the vision sense of the agent. An output node might be linked to this node and result in the action "step-right". The degree to which "wall-left" is true affects the degree to which the action "step-right" is carried out.

3.4 Motion Tree

The motion tree makes use of motion capture clips, which are linked together in a similar manner to a Finite State Machine. Each state or node is a motion clip with edges representing possible transitions between clips. The motion clips are made up of significant actions, such as falling or swinging an axe, as well as smaller transitioning clips which allow an agent to change actions at different points. Motion blending is also used to combine motions. Because the values sent from the brain are based on fuzzy values – not just on or off as with boolean values – Massive agents may effectively combine *actions* as well as behaviours.

Motion blending gives Massive agents the flexibility to interpolate actions for which there is no explicit motion clip. Terrain adaptation is a good example of this. For a particular agent, locomotion cycles are produced for three cases: level terrain, max uphill and max downhill. Massive's motion blending system is then able to produce motion for any slope between the two extremes by blending the level motion with the extreme slope motion to differing degrees [ABL04].

Variation can also be introduced to agents by capturing different versions of the same motion and then blending them together in differing amounts. Agents can even be parameterised with such physical attributes as arm and leg length, as well as more abstract attributes such as aggression. The make-up of a particular agent's body and related motion tree may be randomised to provide further variation.

3.5 Control

Crowds in Massive may be controlled in a number of ways. The most direct and low level method is to change the internal brain variables for an individual agent. Brain nodes are equipped with toggles that allow an animator or director to change the values of particular nodes on the fly [Dun02]. Doing this for multiple agents on a larger scale is likely possible. This would result in the action of a shot being scripted with agents having behaviours turned on or off and altered according to the director's needs. This control is on the level of an agent's behaviour - the agents still react as they see fit. The low level control of an agent's every action is also available to the director, although this becomes unwieldy when considering a large number of agents.

Another widely used form of control (also used by other crowd systems) is flow fields or paths, often used to guide crowds of agents. The implementation of these flow fields is slightly different from those associated with particle crowd models in that the agents can sense the field and choose to follow it. The behaviour of actually following the field must be included in the agent's brain. This is important in that it gives the creator of an agent the flexibility to ignore the force field if something more pressing, such as an enemy, comes along. This control is useful in directing the general flow of the crowd but it is somewhat imprecise. If the agents are not following the field in the correct manner, the animator must resort to manually altering the agents' brains.

Despite these control mechanisms built into Massive, precise control of the crowd simulation is still lacking. Peter Jackson, talking about the use of Massive in *The Lord of the Rings*, jokes about the unpredictability of a crowd simulation using agents designed to think for themselves:

“We literally didn’t know what these guys were going to do. I still get a chuckle when I think about one of the first tests, where we had about 2000 foreground guys desperately trying to kill each other; but in the background, about fifty of them had thought better of it and had turned around to flee the battle! I thought, ‘Those are smart guys.’ It was extraordinarily spooky.”

CHAPTER 4

Fuzzy Logic

The brain of a Massive agent makes use of *fuzzy logic* in order to produce behaviour that mimics the nuances of human behaviour. As we shall see, fuzzy logic ties in well with the way that humans reason and so contributes to the ease with which the behaviour of an agent may be specified. We include here a section explaining some of the important aspects of fuzzy logic and introduce the reasoning behind the implementation decisions made in following chapters. The aim of this chapter is to allow the reader to understand how a crowd simulation may make use of fuzzy logic, not to explain fuzzy logic and fuzzy systems in general. This is by no means a complete or mathematically rigorous tutorial on the subject and so the interested reader is referred to Driankov *et al.* [DHR93] for a more complete handling of the subject.

4.1 Fuzzy sets

The idea behind fuzzy logic was first introduced in the field of mathematical set theory by Zadeh [Zad65a] [Zad65b]. Zadeh introduces the *fuzzy set* as a means of representing knowledge. The concept of *fuzziness* is based on the manner in which humans handle and communicate knowledge. The human language is full of descriptive terms which are innately vague. Words without specific values, such as *large, low, heavy, hot*, are commonly used descriptions. People process and make use of these words with little difficulty. In fact, the vagueness of these linguistic values allows a useful measure of descriptive power that allows humans to communicate general ideas. Fuzzy sets are an attempt to represent knowledge without losing this vagueness inherent in language.

In conventional set theory, an element is either in a set or it is not. The membership function of the set has only two possible value: zero or one. Zadeh introduces the idea of a set of which an element may be partially a member – a set with a membership function whose range is the interval $[0, 1]$.

An oft-used example perhaps best illustrates the usefulness of this concept. If one considers all the people in the world one may then attempt to define which of those are considered *tall.* We could attempt to make use of a conventional set, also called a *crisp* set. This set would clearly not include someone with a height of 1.5m and a person who is 2m would be included. The difficulty lies in where to draw the line. Is a person of 1.8m in the set? What about a person of 1.81m? At some point one must define the boundary where a person of a particular height is in the set but a person even 1mm shorter than that is not. A fuzzy set of people who are *tall*, in contrast, allows a person to be a partial member of the set. A person who is 1.8m tall might have a membership value of 0.6, for example. A comparison of the two possible membership functions is given in figure 4.1.

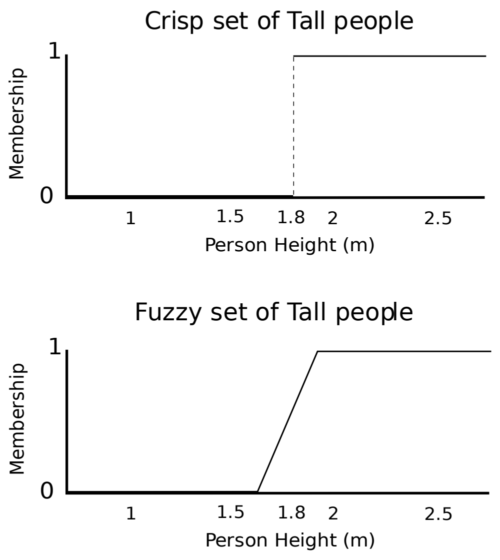


Figure 4.1: *A comparison of the membership functions for the set of "tall" people. The membership function above is for a conventional set, while the membership function below is for a fuzzy set.*

At this point a clarification is required. The fuzzy set, *tall*, attempts to model the vagueness in a single person's understanding of the description "tall". This theory does not account for the varying understandings of the description, that is, the *subjectivity* of the description. The understanding of "tall" may vary widely based on where in the world the description is made. A Swede might consider a tall Italian to not be tall at all, for example. This difference is accounted for by the fact that the membership curve may be defined differently based on the context. The Swede might use a curve that is shifted to the right, for instance. This observation is of importance later when we modify the behaviour of agents.

The membership curves used to define a fuzzy set are theoretically arbitrary. The only limitation is the manner in which the membership curve maps values into the fuzzy set – this should make sense based on the application. In his original papers, Zadeh suggests a number of membership curves that behave well and are widely used. The four that we make use of are shown in figure 4.2. All of these curves have a single peak – the function does not decrease until it reaches its maximum and after that does not increase again. This is of importance in the defuzzification methods discussed below.

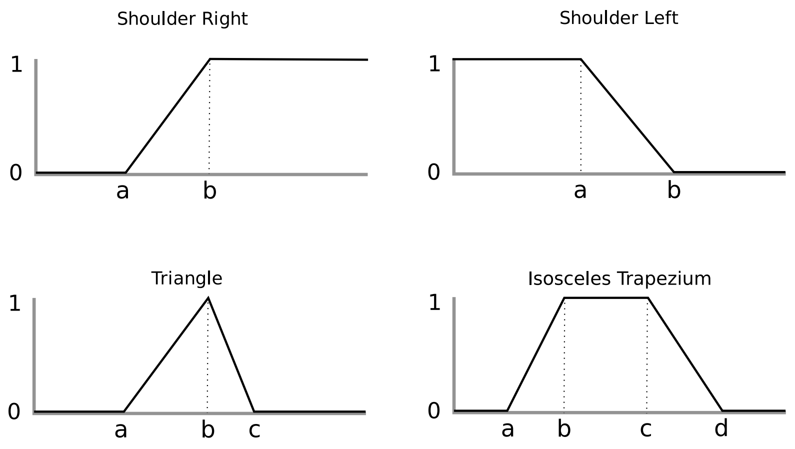


Figure 4.2: *The four different membership functions used by the system.*

An additional linguistic concept that is often used with fuzzy sets is that of a *hedge*. Examples of this concept are the words “very”, “quite”, “somewhat” and “particularly”. In a linguistic sense, the use of these words in a description serves to sharpen or soften the linguistic value to which it is linked. “Very low” is a more severe form of “low”, while “somewhat heavy” is a less intensive form of the value “heavy”. An interpretation of these hedges in fuzzy sets is given by raising the membership function to an appropriate power. Since a membership function returns a value in the range $[0, 1]$, raising it to a power greater than one decreases the value returned by the hedged function. This reduces the fuzzy set represented by the linguistic value and so makes the description more specific. A power less than one performs the opposite task of enlarging the fuzzy set and therefore softening or generalising the linguistic value. For example, if the set *tall*, described above, has a membership function m_{tall}, the membership function of the sets "very tall" and "somewhat tall" might be

$$m_{verytall}(p) = m_{tall}(p)^2 \quad and \quad m_{somewhattall}(p) = m_{tall}(p)^{\frac{1}{2}}$$

for some person, p.

4.2 Fuzzy Controllers

Fuzzy logic is developed to reason with the particular form of knowledge expressed in a linguistic or verbal form. We are interested in the use of fuzzy logic rules by fuzzy controllers as an agent's brain can be regarded as a type of fuzzy controller. The basic make-up of a fuzzy controller is shown in figure 4.3. Crisp inputs are fuzzified by means of the membership functions of fuzzy sets that represent linguistic values. These linguistic values are used in the fuzzy if-then rules through which the fuzzy inference takes place. The membership values are combined in the defuzzification process to give the crisp output values. We now describe this process in more detail.

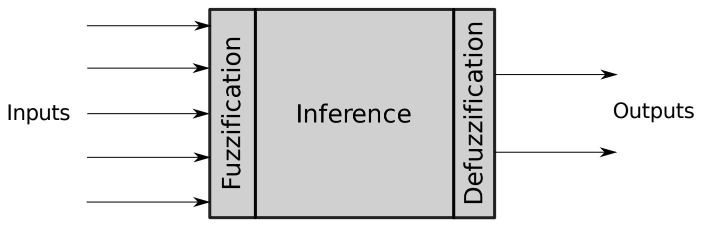


Figure 4.3: *A fuzzy logic controller.*

4.2.1 Fuzzy Inference

A fuzzy controller makes use of a form of fuzzy logic referred to as approximate reasoning. This reasoning takes the form of inference rules that involve fuzzy propositions of the form "X is A". These propositions are similar to those in classical logic with a few clear differences. Firstly, where as a classical proposition is a boolean value, the evaluation of a fuzzy proposition gives a value in the range $[0, 1]$.

Secondly, a fuzzy proposition is made using a linguistic value made up of words or sentences in a natural or artificial language, most often English. An example given originally by Zadeh is of the values of the variable *age*. *Age* may take on linguistic values such as *young, quite young, not young, very young, old, very old*, etc. An example of a fuzzy proposition demonstrating its use of linguistic values is "The room's temperature is hot". Here "The room's temperature" is the variable with the proposed value "hot".

Linked to the use of linguistic values is the manner in which meaning is attached to a fuzzy proposition. Each linguistic value is a fuzzy set, defined by an associated membership function whose domain is the set of values that the variable in question may take on. In the example of the variable *age*, this domain might be the integer values between 0 and 100. Given a fuzzy proposition and crisp value for the variable, the truth, or value, of the proposition may be determined by evaluating the particular value's membership of the linguistic variable's fuzzy set. For example, if the fuzzy proposition is "John's age is young" and the actual value of John's age is 22, the membership function of "young" might map 22 to 0.2. In this case, the fuzzy proposition evaluates to 0.2.

The fuzzy propositions described above are referred to as *atomic* fuzzy propositions. As in classical logic, one can use connectives such as "and", "or" and "not" to create *compound* fuzzy propositions of greater complexity. These compound fuzzy propositions may be further compounded using the same connectives. Examples of compound fuzzy propositions are:

X is A *and* Y is B
X is A *or* Y is B
X is *not* A
not(X is A *and* Y is B) *or* X is B

The meaning of these compound propositions may be found in a similar way to boolean logic, where the truth of a compound statement may be determined by the truth of its components. All that remains is to define the manner in which the connectives combine the truth of the component propositions. There are, in fact, a number of ways in which to do this. We describe and implement the method proposed by Zadeh in his initial generalisation of set theory as it is widely recognised and simple.

For variables x and y, and fuzzy sets A and B with associated membership functions m_A and m_B, the definitions of the logical connectives are as follows:

$$x \text{ is } A \textit{ and } y \text{ is } B = min(m_A(x), m_B(y))$$
$$x \text{ is } A \textit{ or } y \text{ is } B = max(m_A(x), m_B(y))$$
$$x \text{ is } \textit{not } A = not(x \text{ is } A) = 1 - m_A(x)$$

The inference rules of approximate reasoning that make use of these fuzzy propositions are expressed as fuzzy if-then rules analogous to boolean if-then rules. The symbolic expression of a fuzzy if-then rule is

$$if < \text{fuzzy proposition} > then < \text{fuzzy proposition} > .$$

The first fuzzy proposition, referred to as the *antecedent*, determines the strength with which the rule fires. The second fuzzy proposition, the *consequent*, is the implication of the rule's firing. The fuzzy nature of the antecedent means that a rule may fire with differing strengths. As the consequent is a fuzzy proposition, it is not immediately obvious how this implication should be carried out.

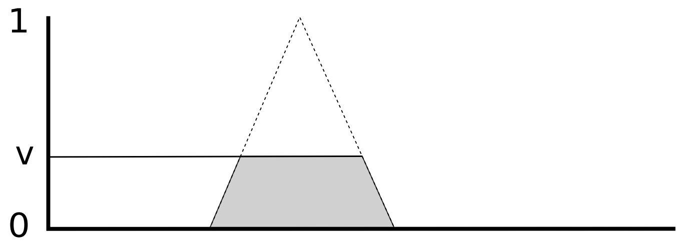


Figure 4.4: *The Mamdani implication caps the membership function of the consequent fuzzy set with the firing strength of the antecedent shown as v.*

There is more than one method of representing the meaning of a fuzzy if-then rule. In fuzzy control the most important is called the Mamdani implication. This is the type of inference that we shall deal with due to its wide-use and simple implementation. For the rule

$$\text{if } x \text{ is } A \text{ then } y \text{ is } B,$$

the Mamdani implication results in the membership function of B being capped by the firing strength of the antecedent as shown in figure 4.4. This modified membership function represents the fuzzy value assigned to the variable y. If the membership function of A is denoted m_A and that of B denoted m_B, then the modified membership function, m_{modB} is given as,

$$m_{modB}(u) = min(m_A(x), m_B(u))$$

for some u in the domain of m_B. The result of a fuzzy rule firing is thus a capped fuzzy set which covers a range of values. This form of fuzzy value is not useful as an output from a fuzzy controller as a change to the system under control usually requires a crisp value. The setting of the fan speed in an air conditioner or the rotation of a servo motor, for example, requires a single real value. Additionally, there are often several rules that are evaluated simultaneously. These rules may assign fuzzy values to a number of output variables with more than one rule assigning values to a particular output variable. In order to combine the effect of rules that assign values to the same variable and determine the crisp value needed to implement the control action, an additional *defuzzification* step is needed.

4.2.2 Defuzzification

Defuzzification is the process of combining the results of fuzzy inference in order to determine a crisp output value. For a particular output variable, v_{out}, the n fuzzy values produced by the fuzzy inference process involving v_{out} are combined by taking their union. Taking the capped membership functions $m_1..m_n$, a new combined membership function m_{comp} is created which is the maximum of the individual functions as follows:

$$m_{comp}(u) = max(m_1(u), m_2(u), ..., m_n(u))$$

for some u in the domain of v_{out}. A defuzzification method is then used to calculate the value of the output variable from this combined fuzzy value. We describe the two most well known methods as well as a third, more robust, method that we used in our crowd implementation.

The *Centre of Area* method [DHR93], shown in figure 4.5, is the best known of the defuzzification methods. Since the control variables used in our crowd system are discrete, we present the discrete version of the method, though the continuous version is similar. For a control variable which takes on a value from the universe $U = \{u_1, u_2, ..., u_l\}$,

$$v_{out} = \frac{\sum_{i=1}^{l} u_i.m_{comp}(u_i)}{\sum_{i=1}^{l} m_{comp}(u_i)}.$$

This method takes into account the whole of the area under the membership curve and finds the value which is in the centre of its area. This method requires the calculation of the area under the curve which is a slow operation, especially when it must be executed a number of times for each agent on every update step. The advantage of this method, however, is that its output changes continuously – a small change in the inputs tends to result in a small change in the output.

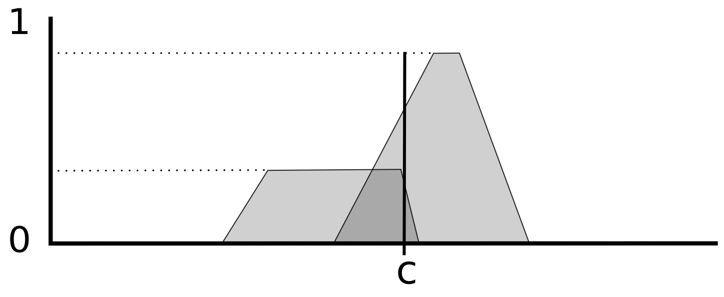


Figure 4.5: *The Centre of Area method.*

The *Middle of Maxima* method [DHR93] is much quicker to calculate but ignores the general shape of the combined fuzzy sets. Instead this method considers only the values of u for which the combined membership curve takes on its maximum value. More precisely, the smallest and largest values of u for which m_{comp} is maximum are averaged as shown in figure 4.6. If the maximum value attained by the combined membership function is given by h, then the value of the output is calculated as

$$v_{out} = \frac{min(u \in U | m_{comp}(u) = h) + max(u \in U | m_{comp}(u) = h)}{2}.$$

The advantage of this method is its computational efficiency. It is also able to handle the situation where the combined membership curve has more than one distinct peak in a more reasonable manner than the Centre of Area method. The main disadvantage is the discontinuous output that may result when the firing strength of two rules is similar.

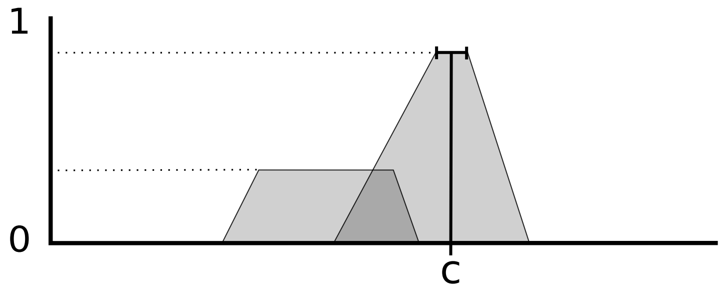


Figure 4.6: *The Middle of Maxima method.*

The last method that we describe is designed to adapt the Centre of Area method to deal with the case of a combined membership curve with more than one peak without losing the continuous behaviour it exhibits. The *Centre of Largest Area* method [DHR93], shown in figure 4.7, finds the largest connected subset of the area under the membership curve and then determines the centre of its area.

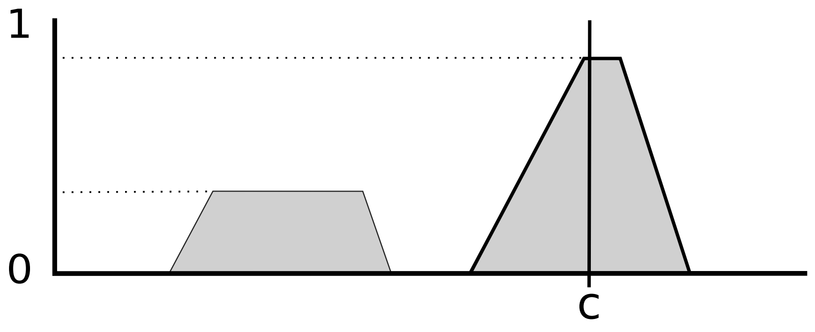


Figure 4.7: *The Centre of Largest Area method.*

Fuzzy logic is well suited to the modelling of intelligent behaviour in crowd members because it is based on how humans think and handle knowledge. This property provides two distinct advantages. Firstly, the agents are more likely to behave like intelligent humans if their thought process is similar. Secondly, the linguistic basis of fuzzy logic is far removed from the differential equations and other analytical methods for describing complex systems. This allows a non-technical user to be able to specify the thinking of an agent, and hence the behaviour of a crowd, by simply applying the rules that he or she might follow. There is a close link between how the user thinks and how the agent thinks which offers an advantage in authoring the crowd.

CHAPTER 5

Design

The work presented in this dissertation is concerned with the lack of control provided by current crowd research. We propose constraint-based control over the behaviour of a crowd for use in a crowd animation system. This section details the design of such a crowd control system.

5.1 High Level Control

There has been some work done in the area of crowd animation aimed at adding control. Most crowd animation research takes a bottom up approach by simulating the individuals and control may be provided at this level. This work, discussed in section 2.2.4.1, either attempts to enable the agents to follow a user's direction as part of its behaviour, or allows a user to directly manipulate the individuals involved in the simulation. The former approach requires an additional level of intelligence and so adds significant complexity to the agents. The latter form of control is effective but limited to small scale changes as tweaking many individuals is tedious.

It is more meaningful and effective to take a top-down approach when considering high-level control. Individual intelligence can produce realistic and complex simulated behaviour but global control may require global intelligence. The crowd animation approaches that search the motion graphs of characters in order to exactly meet constraints are a good example of this (see section 2.2.5). Unfortunately, these methods are not ideal for crowd animation as they produce zombie agents with no real behaviour or intelligence. They do, however, effectively provide complete control to the user by means of position constraints that, if achievable, are guaranteed to be met. An additional disadvantage is the complexity of such a search. Although some motion graph methods perform in a reasonable time-frame, this is due to optimisations that are specific to positional constraints.

Control as a form of post process in given is a recent paper by Kwon *et al.* [KLL08]. This approach alters the trajectory information provided by a crowd simulation by linking the positions of neighbouring agents, across all time steps, together in a mesh. Standard mesh deformation techniques are then used to deform and arrange the mesh such that the agents move as desired. This form of control is not applicable to crowds motion that consists of full bodied articulated motion. The trajectories of the agents may be determined from their full motion but the loss of this information seems a high price to pay.

An effective, though rather specific example of top-down control is provided by Anderson *et al.* [AMC03]. The bird agents produced by Reynolds (described in section 2.2.3) are constrained using an approach which effectively samples the space of possible behaviours, accepting only behaviour which result in the constraints being met. The sampled behaviours are evaluated for fitness based on how well they match the behaviour of the unconstrained model. This evaluation makes use of the fact that, in the bird flocking model, the steering behaviour includes a random wander element. If the wander contribution implied by the sampled behaviour appears to have been drawn from the same distribution as the wander contribution of the unconstrained behaviour, the sample receives a high fitness score.

This iterative improvement of the constrained behaviour works well in this case, though for large numbers of birds the system may be somewhat slow. Unfortunately, there are a number of attributes of the flocking behavioural model which the system exploits, making the generalisation of this approach problematic.

In this dissertation we develop a general method of high-level control for agent-based crowds. This method is inspired by previous top-down approaches and is in the form of general constraints. The system we develop attempts to satisfy these constraints by altering the behaviour of the crowd. This is done while trying to retain the behaviour of the original crowd as much as possible.

5.2 The Crowd Simulation

Massive is the state of the art in crowd animation for film and any further work in this area must take it into account. The animation quality produced by Massive has been shown to be of high quality. Its authoring tools are effective and complex behaviour is made possible by its fuzzy logic inference framework. The widespread use of Massive shows that the computational cost is low enough to make it useful. Since Massive produces such good results in many areas, it would be foolhardy to start from scratch. We choose instead to take Massive as a starting point in our crowd animation work and build on its successes.

The system we create attempts to reproduce the successful principles of the Massive crowd animation system in order to avoid re-inventing the wheel. Reproduction of such a widely used system also makes this research relevant to the visual effects industry. We do, however, make a number of simplifications to the Massive system due to time constraints and in order to simplify the implementation of the system as a whole. As we are attempting to explore the control of a crowd's behaviour, the quality of the motion produced is of little concern. The implementation of a motion tree with motion blending is a considerable task that requires additional resources in the form of motion clips. Since this aspect of Massive is not relevant to the behavioural constraints we wish to place on the crowds, we make the simplification of removing the motion tree. Our agents are able to take simple actions, such as *step forward* or *turn right* without the accompanying articulated motion.

Variation within a crowd is produced in Massive by randomly varying attributes in both the brain and motion tree. This variation increases the realism of the simulation and reduces the visibility of repeated motion clips throughout the crowd. However, many of the crowd simulation systems described in chapter 2 are able to produce emergent crowd behaviour without any variation - their agents are all identical. In order to simplify the crowd simulation, random variation in agents is not implemented. We recognise that random variation may be leveraged by additional crowd control methods but this is beyond the scope of our research (see section 9.2).

5.3 Constraints

A requirement for the design of a system of crowd simulation control is to determine the manner in which the control should be applied. Ideally, a crowd control system should be general, in the sense that any particular aspect of the crowd simulation may be controlled. Building on ideas from previous work in crowd control such as that by Lau and Kuffner [LK06] and Anderson *et al.* [AMC03], we design a method of control by means of constraints. A user may specify constraints on a crowd which the control system then attempts to meet by altering the crowd simulation automatically. These constraints are arbitrary and may be applied to any aspect of the crowd simulation.

This system for the specification and satisfaction of constraints is built on top of the crowd simulation system. The general use-case is described in figure 5.1. An existing simulation, involving any number of groups of agents, has constraints applied to it by the user. These constraints, along with the parameters which describe the simulation, are then fed into the constraint satisfaction system (CSS) which alters the simulation parameters such that the constraints are met.

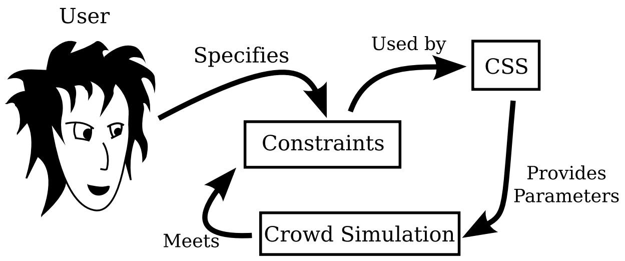


Figure 5.1: *A general use-case of our crowd control system.*

The constraints are not limited and may be applied to any part of the crowd simulation over any time frame. It is likely that some constraints may be partially met and there is the possibility that a sets of constraints cannot be fully met by a particular crowd. In order to allow the system to improve the simulation as much as it can, constraints include a value that ranges from zero to one. If a constraint is completely violated then its score would be zero while a constraint fully met would have a score of one. The introduction of a partially met constraint is a generalisation of the boolean, met or not met, constraint and so does not exclude constraints of boolean form.

In order to simplify the study of this control mechanism, no study is made of how these constraints should be specified or used. The motivation for the usefulness of constraints is convincing – completely general constraints provide unlimited control. However, the manner in which these constraint should be used and specified is beyond the scope of this work. We are attempting to prove the concept of general constraint-based control, not provide a complete study of this this form of control.

5.3.1 Constraint Satisfaction

Once the constraints required by a user have been specified, the CSS is required to provide the user with a crowd simulation which meets the particular constraints. The first question that must be answered before such a system can be designed is that of what simulation parameters the CSS should be allowed to change. Previous work in crowd animation generally changes the motion of the individuals. Systems that deal with particle motion, such as the work in motion editing by Kwon *et al.*[KLL08], alter the trajectories of the particles in order to have them meet the constraints. The work on searching motion trees in order to meet the requirements of a user by Lau and Kuffner [LK06] and Sung *et al.*[SKG05] set the motion of the agents based on constraints.

This method of constraint satisfaction is severely limiting in that it abandons the behaviour of the agents and only allows control of an agent's motion. The altered bird trajectories of Anderson et al. [AMC03] attempt to satisfy the agent's original behaviour but this approach still suffers from lack of generality in the constraints. As mentioned previously, these constraints are tied to the agent's behaviour by making use of the random wandering of a bird. This specific property does not exist in more general crowd models.

We propose that the ideal way in which to meet the constraints of a user is through the agent's own behaviour. This approach is consistent with the behavioural animation used to animate crowds. Instead of ignoring the behavioural model and dealing only with its motion output, as described above, it is advantageous to rather improve this model. By changing the underlying behavioural model, the new crowd may be used in different situations and even changed using existing control methods.

An additional advantage in meeting crowd level constraints by changing the behaviour of the crowd is that this allows constraints to be specified, not only with reference to the motion of the agents, but also with reference to their behaviour. If only the motion of an agent is available to be constrained, then the physical outcome of its deliberation process may be controlled. If the properties of an agent's brain are also part of a constraint then greater control of an agent's behaviour is achieved. Although we do not actively explore the avenue of how constraints may be used, this flexibility seems desirable.

At this point a clear distinction must be made between the control that we propose and the behaviour based control provided by the approaches such as the cognitive modelling of Funge *et al.* [FTT99]. Whereas cognitive based control attempts to include the agent's own intelligence in the constraint satisfaction process, we satisfy these constraints in a global, top-down manner. Additional deliberation may enable an agent to figure out the correct actions to meet user constraints, but we propose a system which changes the agents' behaviour so that they naturally meet the constraints, acting without any knowledge of the constraints themselves.

Having established that the behaviour of the agents in a crowd should be changed, the manner in which this should be done is still undecided. There are a number of different options and some of the alternatives to our chosen method may prove fruitful (see further discussion in section 9.2). We make use of the impressive crowd simulations produced by Massive as a starting point for our work. Therefore, the parameters of the crowd simulation that we choose to alter are based on Massive's implementation of an agent's behaviour using fuzzy logic. Since Massive creates agents as instances of a master agent, we choose to alter the brain parameters of the master agent.

The advantage of altering the behaviour of the master agent, and hence altering the behaviour of a whole group of agents, is that our system then adds an additional layer of control without removing the existing methods. The brains that are altered by the CSS are still available to be manually edited by the user and to be used in different scenarios. In this way, the control provided by our constraint system supports the authoring of crowds. The disadvantage of changing the behaviour of a whole group of agents is that there is less power available in the system as the same changes must be made to all instances of the master agent. However, a large amount of the control available to a Massive user in creating a crowd is through the direct editing of the master agent's brain. We are simply providing a method to make these changes automatically. In addition, techniques that alter individual agents (see section 9.2) may be used in addition to our system.

As described previously, the brain of a Massive agent makes use of fuzzy logic. One of the interesting interpretations of the membership function of a fuzzy set is that it gives a person or group of people's understanding or interpretation of a fuzzy concept. For example, a person is asked to define a fuzzy set "comfortable" over the universe of possible room temperatures as shown in figure 5.2. A person from Iceland might provide the curve shown on the left while a Jamaican might give the function shown on the right.

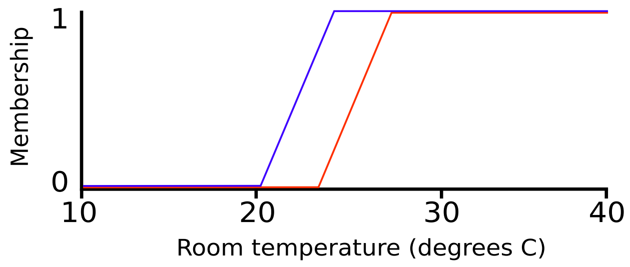


Figure 5.2: *Two different definitions of a "comfortable" temperature.*

This same observation may be applied to the agents in a Massive crowd. The crisp values which an agent is provided with by its senses are converted to membership values of fuzzy sets. The membership curves used to provide these values define the agents understanding of the linguistic values that the fuzzy sets represent. The agent's brain makes use of these linguistic values in the reasoning given by fuzzy rules. We argue that by changing the agents understanding of these linguistic values, the agents behaviour may be changed in a manner which remains true to the reasoning that lies behind it. The agent's understanding of linguistic values is given by the membership functions of the particular fuzzy sets and so it is these membership functions that must be changed by our system. To illustrate, in order to change a person's degree of reaction to cold, one might change what he considers cold to be.

It must be noted that the control we propose is a layer above the standard control provided by the crowd simulation system. Once a brain has been altered to satisfy constraints, a user may make use of the standard brain authoring tools in order to further change the master agent's brain. In this way, the constraint-based control is added as an additional form of control and in no way removes the methods of control that exist already. By allowing a user to change brains that have undergone constraint satisfaction, this method of control is useful not only for the per-shot tweaking of agents, but also for the authoring of agents' behaviour in general.

5.4 Testing Strategy

In order to determine the success of the CSS, we produce a strategy for its evaluation. The testing strategy is determined by the requirements of the control and so is given without reference to implementation details. The requirements of control that we propose are as follows:

- **Provide additional control** - Constraints that cannot be met easily through other means should be satisfied by this system.
- **Execute in an acceptable time-frame** - The constraint satisfaction process should take place in a reasonable amount of time.
- **Produce minimal behavioural change** - The crowd behaviour that is produced by the system should as closely match the original crowd as possible. The change should be no more than that produced by an expert user of the system applying the same constraints to the simulation by manually editing the agents.
- **Applicable to real world use** - The test agents and scenarios should be similar to those used in real applications. A variety of representative test cases are necessary.

Based on these requirements, we produce a testing strategy that is made up of a number of test scenarios. Each scenario consists of a crowd of agents, perhaps of differing types, and different environments. These scenarios are based on possible real world examples of where crowd animation systems may be used. A set of constraints that might reasonably be required are placed on the simulation for each scenario. First an expert user, who serves as our control case, attempts to meet the constraints by altering the behaviour of the agents in the simulation. The automatic CSS then attempts to meet the constraints.

Once each scenario has been met by these two different means, the performance of the two may be compared in different ways. The immediate and most important consideration is how well the constraints are met. Since each constraint is structured such that it has a satisfaction value ranging from zero to one, constraints that are not completely met may still be judged based on how well they improve the satisfaction of constraints.

The time taken by the automatic method may also be roughly compared to the time required by the expert user to meet the constraints. This timing comparison is based on only a rough estimation of how the system may be actually used. There are factors, such the time taken to specify the constraints and the fact that with an automatic system the user is free to perform other tasks while the system is running, which are not considered in the comparison. However, this simple comparison of running times gives an idea of the possible usefulness of the CSS.

A third, and important, comparison is based on how much the crowd behaviour is changed when the constraints are satisfied. An automatic system which accurately applies constraints, but does so at the expense of the behaviour which has been carefully crafted, is of little use. Since the similarity of crowd behaviour is a rather subjective, qualitative measure, we perform human experiments that compare the crowd behaviour of the altered crowd to the original. Subjects are required to perform comparisons of short video clips showing crowd animation and assign a similarity score to the behaviour of the crowd in the different clips. The video clips are shown in pairs with the first clip showing the original crowd behaviour and the second the crowd altered by either the expert user or the CSS. By requiring users to perform a number of such comparisons, similarity scores for the two different constraint satisfaction methods may be gathered and compared. This gives an indication of whether the automatic CSS alters a crowd's behaviour more than is necessary.

5.5 Design Summary

In this chapter we have described the design of a system which adds needed control to current crowd simulation systems. The design decisions are summarised as follows:

Built on top of an existing crowd simulation system
The crowd simulations to which control is applied is based on the widely used Massive platform.

Control by means of constraints
Control of the crowd simulation is achieved through the specification of general constraints. These constraints describe the outcome that the user desires. The control system alters the crowd such that the constraints are met.

Change agent behaviour
The constraints specified by the user are met through altering the behavioural model of the agents involved in the simulation. Changes to the agent brains are made automatically so that their behaviour results in the constraints being met.

CHAPTER 6

Crowd Simulation Implementation

The implementation details of the system that applies the design described in the previous chapter, is broken into two parts. This chapter details the implementation of the the crowd simulation system with the constraint satisfaction system (CSS) described in the following chapter.

The aim of this implementation is to demonstrate the usefulness and potential of constraint-based control. This is done by presenting an example of a crowd simulation system to which this form of control is added. In this implementation, certain decisions were made based on a number of considerations – it is noted where alternative choices may prove fruitful. The aim is to explore the concept of constraint-based control, specifically for behavioural modification of agent-based crowd simulations, and this aim guides the implementation decisions.

Our crowd simulation system is based on the widely used Massive platform. Section 3 gives the information on Massive that can be gleaned from interviews, articles and use of the system. Implementing the principles of that system requires filling in a number of gaps and so it is important to recognise that we do not claim that this system will work in exactly the same way as Massive. We have made simplifying design decisions (detailed in section 5.2) and must also interpret the data that we have available. Inaccuracies are bound to occur and all we claim to produce is a system that is based on the successful principles applied by Massive.

6.1 System Overview - The world

The crowd simulation system is implemented in C++ in an object-oriented manner. This system, shown in figure 6.1, is based on a global *world* object. This world contains the agents, environment and any other miscellaneous information required by a simulation. At the highest level, a simulation involves initialising a world object and then repeatedly calling the world's update method. The simulation uses forward Euler integration for simplicity and so the only information the world requires to update itself is the time elapsed since the last update.

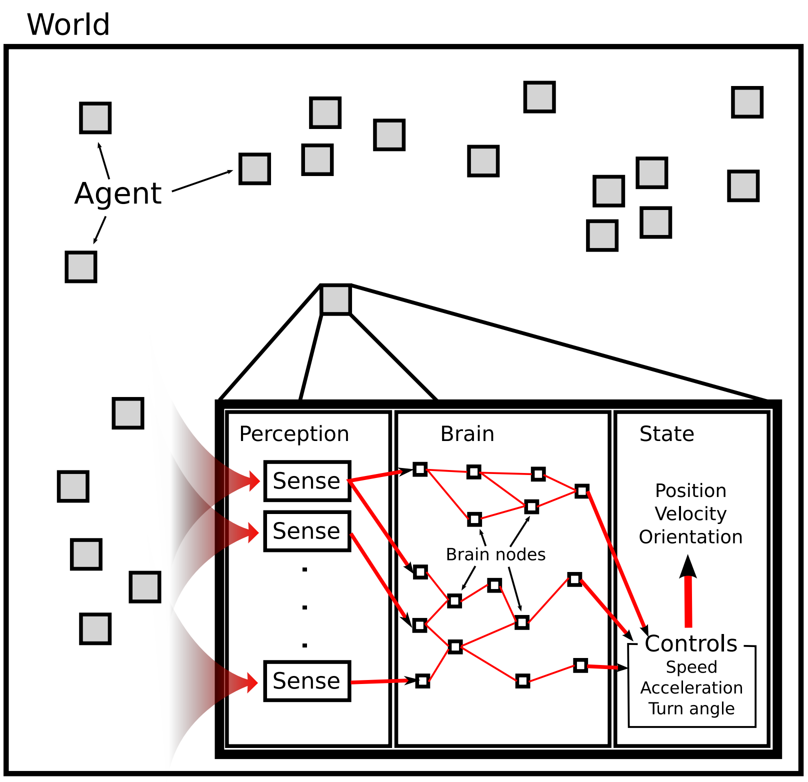


Figure 6.1: *An overview of the crowd simulation system showing the detailed view of a single agent in the world. The flow of information through the agent is shown in red.*

The simulation that is encapsulated by the world object involves a number of *agent* objects along with some environmental geometry specified as two dimensional polygons. These polygons are converted into a 3D environment by placing them on the xy plane and extruding them in the positive z direction. Each line segment of a 2D polygon becomes a rectangle in 3D space. The floor of the environment is considered to be the plane where $z = 0$.

An agent contains a *state* object that describes details such as its position, velocity and orientation. In addition, this object has a number of control variables which may be used to cause the agent to perform certain actions. Along with a state, each agent has its own *brain* and *perception*. The perception object contains a number of *sense* objects, possibly of differing types. These are responsible for querying the world object and so supply information based on which the agents act.

The brain contains any number of *brain node* objects which have different subclasses. Each brain node may be dependent on others and so an ordering is imposed upon them. In this way, they form a directed graph, similar to that of the Massive brains described in section 3. Brain nodes that provide input to the brain are linked to *sense* objects. Other brain nodes update control variables and serve as the brain's method of providing output. The combination of brain nodes represent a number of fuzzy logic rules which define an agent's behaviour.

When the world object is updated, it sequentially calls the update methods of each of the agents in the simulation. Each agent updates it perception, calls the decision function of its brain to update its control variables, and then updates its state according to these controls. This sequential update results in each agent taking a turn to move while the others remain still. Because an agent's behaviour is independent and the simulation models a system which is, in fact, continuously changing, this detail makes little difference to the results produced.

6.2 Agents

From an external point of view, an agent is defined by its state. While an agent's behaviour attempts to update this state, the virtual reality of the agent is encapsulated in its state object. Our implementation differs from Massive in that we do not give the agents a motion tree with complex articulated bodies. This significantly simplifies an agent's state and action implementation.

The crowd model that we implement makes a clear distinction between the action attempted by an agent and the effect that this attempt has on the agent's state. While an agent might want to move in a certain way or accelerate in a particular direction, properties of the world may not allow it. This separation allows physical simulation to be included in the animation of the crowd, if desired. We define an agent's virtual reality, which we refer to as its physical state, using state variables included in the state object. The intended actions are defined by an agent's control variables, also included in the state object.

Due to our focus on the behaviour of an agent as opposed to the quality of its motion, we make use of embodied agents with only simple, non-articulated bodies. The state of an agent's body in the 3D virtual space is therefore uniquely defined by a 3D position vector and two orientation vectors. The two orientation vectors define an agent's "look" direction and "up" direction as shown in figure 6.2. In addition, we maintain a third, orthogonal, orientation vector defining the agent's "right". This vector is used in calculating angles for the hearing sense (see section 6.4).

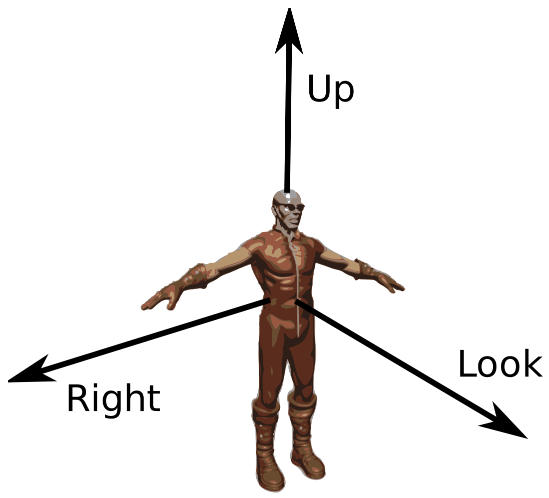


Figure 6.2: *The orientation vectors of an agent.*

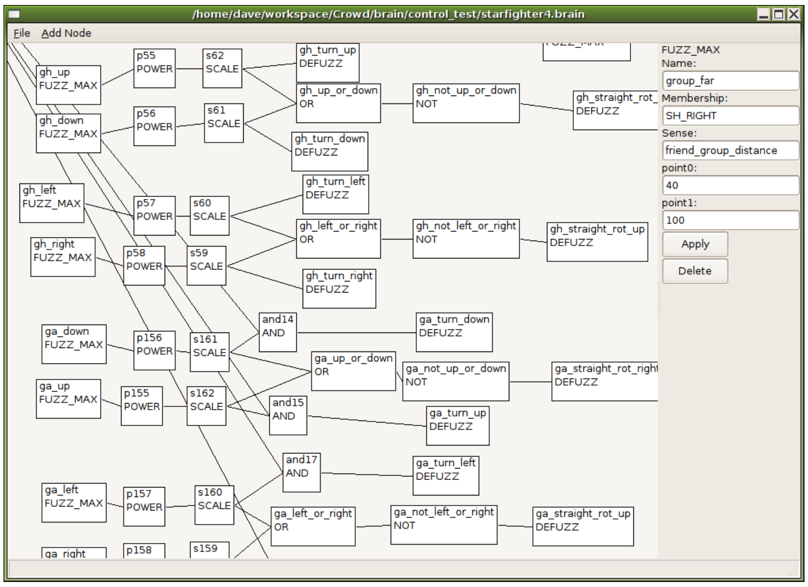
An agent's physical state is set by the variables described above and is the same for all agents. The manner in which an agent may attempt to update these variables varies from one agent type to another. An agent's actions are defined by a number of discrete valued control variables. Each control value has a minimum and maximum value as well as a step value which defines its granularity. A particular agent type may have any number of different controls which dictate how they are used to update an agent's state. A car agent, for example, might have an *acceleration* control, a *speed* control and a *turn angle* control. A killer robot might include a *shoot* control and a *dying* control. The controls implemented in the testing of our system are detailed in section 8.2.

The controls of an agent are linked to the brain's output nodes. When an agent is updated, the brain supplies new values for these controls (assuming that the brain's behaviour affects the particular control). The agent's state is then updated according to the control values. The effect of the control values differ and new controls with associated effects may be added. For example, the *acceleration* control changes the magnitude of an agent's velocity vector while the *turn around up* rotates an agent's look and right orientation vectors about the up orientation vector. This separate update step allows the agent's attempted action to fail. The *shoot* control, for instance, does not cause the agent to fire its weapon until the weapon's cool-down period has elapsed.

6.3 Brains

The brain is the centre of an agent's behaviour and so is the area in which much of our interest lies. The brains of Massive agents make use of fuzzy logic to combine the inputs from an agent's sensors and determine the actions it should take. This process makes use of fuzzy rules, represented as a graph of brain nodes. Since the details of the Massive implementation of fuzzy logic are not available, the literature on fuzzy system control is an importance source. In some areas, particularly with reference to the brain inputs, Massive differs from the standard model of fuzzy control. Our system follows the example of Massive in the cases where the reasoning behind such deviation is known.

In our implementation, the brain is made up of a number of brain nodes. These nodes, with the exception of output nodes, may be though of as fuzzy values and so have a value in the range $[0, 1]$. Each node may also have a number of nodes upon which it depends. These nodes must be evaluated before the dependent node and, when updated, the dependent node is sent values of the nodes upon which it depends. Each node may be represented as a node in a graph with dependencies creating edges between them as shown in figure 6.3. The eight different node types are described below.



FUZZ_MAX
Name: group_far
Membership: SH_RIGHT
Sense: friend_group_distance
point0: 40
point1: 100

Figure 6.3: *The graph structure of a brain shown in our brain editing tool.*

6.3.1 The Fuzz node

This node is one of two input nodes which convert the information sent to the brain from the agent's senses into a fuzzy set membership value. Massive provides its agents with sense information in some form of real valued array – differing from standard fuzzy controllers in which a single real value is the input. As an array input provides the flexibility required for agent vision, we follow the lead of Massive and design the senses to provide their information in the form of a data array with each array element a floating point number.

In Massive, the values of this array are converted to a fuzzy value by means of a membership curve that is super-imposed over the array as shown in in figure 6.4. Our system implements this fuzzification using a membership function, f, derived from this curve that has a domain running the length of the data array and a range that is from zero to one. For a data array, $\mathbf{d}$, of length n and with maximum value m, the membership value is calculated as

$$\frac{\sum_{i=1}^{n} (d_i \cdot f(i))}{\sum_{i=1}^{n} (m \cdot f(i))}.$$

This averages the values of the sense's data array while giving greater importance to the parts of the array for which the membership function has high value. The parts of the array for which the membership function is zero have no effect at all on the calculated value.

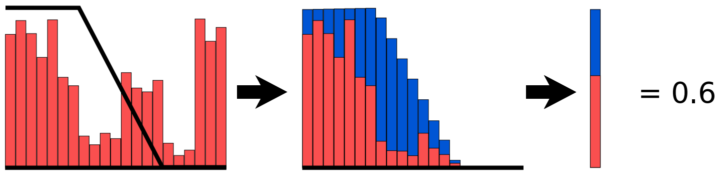


Figure 6.4: *The use of a membership function by the Fuzz node to fuzzify the data array supplied by an agent's sense. The fuzzification effectively divides the scaled "sensed" area by the area under the curve, providing a value between zero and one.*

There are four different types of membership curves used by the crowd simulation system. These are shown in figure 4.2 and are named *shoulder left*, *triangle*, *shoulder right* and *isosceles trapezium*. The choice of these functions, although slightly different to Massive, which uses smoother curves, is based on their widespread use in fuzzy control. The Fuzz node receives input from the senses and therefore depends on no other nodes.

6.3.2 The Fuzz-Max Node

The second input node is similar to the *Fuzz node* in that is makes use of a membership curve to convert the data array, supplied by a sense, into a fuzzy value. The difference is that the averaging effect is removed. The Fuzz-Max node was developed to allow for senses which provide exact information such as the *target up angle* sense described in section 6.4.

The approach of the Fuzz-Max node is to recognise only the strongest signal with reference to the membership function, f. For a data array, $\mathbf{d}$, of length n and with maximum value m, the membership value is calculated as

$$max_{i=1}^{n}\left(\frac{d_i \cdot f(i)}{m}\right).$$

In this calculation, high values in the data array as well as values where the membership curve is close to one are favoured. The Fuzz-Max node, like the Fuzz node, does not depend on any other nodes.

6.3.3 The Defuzz Node

The Defuzz node has the task of converting the result of the inference process into a number that is useful to the agent. Each Defuzz node provides a value for one control variable though a particular control variable may have more than one Defuzz node which sets its value. The node has one dependent which provides the fuzzy membership value that needs to be defuzzified. The full defuzzification process involves not only converting the values from the brain's fuzzy variables into crisp, control values, but also combining these control values. The combination step is described more fully in section 6.3.7. Here we describe the work of the individual defuzz node.

A Defuzz node makes use of a membership function which describes a fuzzy set defined over the possible values of its particular control variable. The functions used by the system are the same ones used by the Fuzz and Fuzz-Max nodes (shown in figure 4.2). The inverse of this function is used to map the value of the fuzzy variable to a value of the control variable. Since the inverse of the membership function is not one-to-one, the inverse mapping is likely to provide a range of values. The selection of a specific value is part of the combination process which combines all the Defuzz nodes for a particular control variable.

6.3.4 Logical Connective Nodes

There are three logical connective nodes: AND, OR and NOT. These nodes implement the fuzzy connectives described in chapter 4. We implement the min/max set of connectives because of their simplicity, efficiency and wide-use.

Both the AND and OR nodes combine the values of two other nodes. The value of the fuzzy variable represented by the AND node is determined by combining the values these two nodes by taking their minimum. Intuitively, the AND node describes the degree to which two different statements are true: both of them are only as true as the least true one. Where v_1 and v_2 are the two incoming values,

$$v_{AND} = min(v_1, v_2).$$

The value of an OR node is calculated by taking the maximum of the two values, intuitively describing the degree to which either one is true. Where the values of the nodes on which the OR node depends are v_1 and v_2,

$$v_{OR} = max(v_1, v_2).$$

The AND and OR nodes may be combined in a manner similar to that of their boolean logic counterparts and similar identities exist to boolean logic.

The NOT node provides the opposite of a particular node. To this end, a NOT node depends on a single node and has a value which, from a fuzzy inference point of view, is opposite to that of the original node. Since a fuzzy variable must have a value in the range $[0, 1]$, this compliment is given by

$$v_{NOT} = 1 - v_{orig}$$

where v_{orig} is the value of the original node.

6.3.5 The Scale Node

Both the Scale Node and Power Node (described below) are linked to the concept of *hedges* from fuzzy set theory, described in section 4. Scaling is not the way in which hedges are usually described in fuzzy control theory. We find Scale nodes useful, however, in reducing the strength of a particular input or creating a behaviour which is easily overridden by other behaviours. It is likely that these effects could be accomplished using a combination of other nodes, albeit in a more involved construction.

A scale node simply scales the value of a particular fuzzy variable. It has one node upon which it depends and a scale factor associated with it. For an input value of v and a scale factor of s, the value of the Scale node is simply

$$v_{SCALE} = v \cdot s.$$

6.3.6 The Power Node

The Power node is a truer implementation of the hedges referred to above. A Power node serves to intensify or soften the value of a particular fuzzy variable. The node has a power attribute and depends on one other node. The value of the Power node, v_{POWER}, with power attribute, p, and input value, v, is given by

$$v_{POWER} = v^p.$$

If the value of the power attribute is greater than one, the Power node serves to reduce the strength of the variable. If the power is less than one, the variable's value is magnified.

6.3.7 Defuzzification

Updating a brain is equivalent to the evaluation of the fuzzy logic rules that is contains. This is done on a node by node basis. The brain simply loops through the array of brain node objects, supplying each with the values of the nodes upon which it depends (if any) and calling the particular node object's update routine. The evaluation of a node before its dependant is ensured by the brain authoring tool (see section 6.5 for a description of the tool). This tool saves the brain nodes in order of dependence so that the array of brain nodes need only be updated in sequence and a dependent node will always be updated after the nodes on which it depends. Circular dependencies are detected by the brain editor and the user notified as a valid set of fuzzy rules does not produce such a problem.

The input nodes of the brain contain pointers to their relevant sense objects. An agent's perception, containing all of its sense objects, is updated prior to the brain, ensuring that an agent obtains up-to-date sensory information. The evaluation of each node completes the fuzzy inference. All that is left is to combine the outputs in order to determine the values of the agent's control variables. This is the defuzzification step.

Defuzzification is done on a per-control basis by looping through the agent's control variables. At an agent's initialisation, the indices of all the defuzz nodes that affect a particular control are entered into the control's defuzz index array.

This allows direct access to the correct defuzz nodes when calculating the new value for a control. There is likely to be more than one defuzz node which affects a particular control. Each one will have its own associated membership function and hence a range of values that the control could take.

A defuzz node has its membership function capped by the its membership value – the value of the brain node on which it depends. Any values of the membership function which are above the node's value are reduced to this maximum as shown in figure 4.4. The membership functions of the different defuzz nodes are combined by taking the maximum of each of them over the domain of the control as described in chapter 4. This combined and capped membership function is used to determine the value of the control.

We initially implemented two of the most well-known defuzzification method (described in section 4). Both of these methods exhibited undesirable effects in different situations. We first implemented the Centre of Area (COA) method but found that it caused agents to crash into world geometry.

An example of where this occurs is given by a car approaching a wall at near a right angle to the wall, as shown in figure 6.5. The car agent has two fairly symmetrical steering rules which are applied to a turning control. One of these causes the car to turn to the left when there is an obstacle near on the right, and another to turn to the right when there is an obstacle near and to the left. The defuzz nodes associated with these behaviours are shown in the figure. Since the information about the left and right obstructions is likely to be similar, the two behaviours will fire with similar strength resulting in the combined membership function show. Taking the centroid of this combined function results in a turn value of close to zero. The car's behaviour drives it straight into the wall.

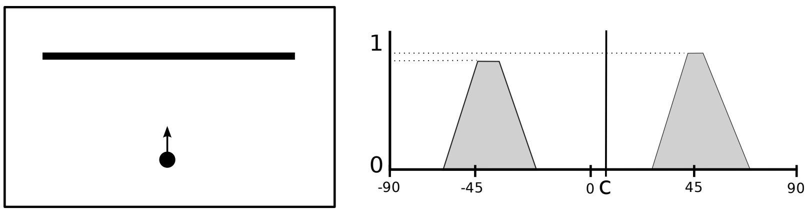


Figure 6.5: *The problem with the Centroid of Area defuzz method.*

This problem came up frequently in our experiments and it is difficult to solve, even with careful specification of brain rules. The COA method leaves the agent unable to choose one course of action if there are two conflicting ones of similar strength. The agent attempts to do both and so ends up doing nothing. The cause of this problem is the manner in which the defuzzification process deals with the multiple peaks of the combined membership curves.

A defuzzification method that does not suffer from this weakness is the Middle of Maxima (MOM) method. Although the implementation of this method solved the problem described above, we found that agents movements were jerky, with the control values oscillating rapidly. This undesirable effect is due to the MOM method acting based on only one rule's result and ignoring the rest as shown in figure 6.6. The two cases shown are based on only a small change in input values, which results in a small change in the firing strength of the two rules. However, this causes the one rule to fire with slightly greater strength than the other and so has the undesirable effect of moving the control value by a large amount.

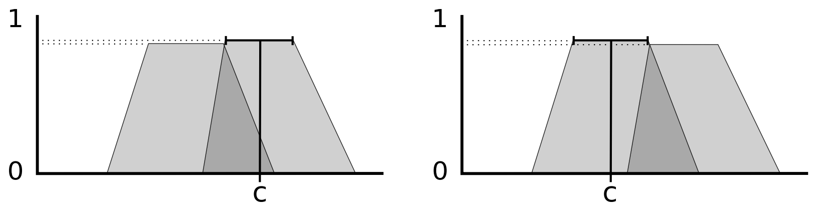


Figure 6.6: *The problem with the Middle of Maxima defuzz method. A small change in the input values results in a large change in the value of the control output, c.*

The defuzzification method which overcomes these problems is the Centre of Largest Area method (see section 4.2.2). The connected areas under the curve are examined and the largest selected. The centre may then be calculated in a similar method to that used for the COA method. This defuzzification method causes an agent to choose one of the options when there are distinctly different possibilities for the control value and yet changes the control variable in a continuous manner.

6.4 Senses

The brain of an agent alters an agent's control variables based on the combination of its inputs using fuzzy inference. These inputs are provided by an agent's senses. The sense object is a general interface which may gather information from any source in the simulation's world object, including the agent to which the sense belongs. To this end, the world object is provided to the sense when it is called on to update itself. An important part of Massive's agent architecture is the synthetic vision with which agents are provided. As this sense is the most complicated, we allow the implementation of synthetic vision to guide the design of the sense interface as a whole.

6.4.1 Synthetic vision

As more fully described in chapter 3, one form of input that Massive agents are given to their deliberation process is a form of synthetic vision. We know that this vision is a simplified 2D scanline rendering of the 3D scene that the agent inhabits with other agents and objects rendered simply. From using Massive, it can be seen that the colours of agents may be rendered according to their type and an agent told to only notice certain colours.

The fuzzy inference that takes place in the brain makes use of fuzzy variables which take on a value in the range $[0, 1]$. It remains to determine how the 2D image provided by the vision sense can be converted into a real value in this range – this is not entirely obvious from using Massive. There is the use of some form of membership curve, similar in appearance to the membership curve of a fuzzy set. How this 2D curve is used to fuzzify the 2D image is, however, unclear. There are also at least two different types of vision input available to an agent: "vision x" and "vision y". These seem to give 2D images that present what an agent sees with reference to its x and y axes.

In our implementation we deal with this problem in two steps. We force a sense to provide its data to the brain in the form of an array of real values with some specified maximum. This data array is then converted to a real number by using a membership curve. This second step was described in section 6.3.

We follow the example of Massive in providing two different vision senses, one for each of the x and y axes. We discuss the case for the x axis shown in figure 6.7. The case for the y axis simply swaps the use of the x and y axes. In either case, the world is rendered from the agent's point of view at low resolution, typical values being 100 by 70 pixels. The objects in the world are rendered without lighting and every object has a simplified model used to render it in agent vision. This was not really necessary in our case as we have no high resolution models. When using the detailed models required by visual effects, however, efficiency dictates the use of less costly alternatives.

The 2D rendered vision is converted into a data array of length equal to the x resolution of the image. Each element of the array corresponds to an x value in the rendered image. The image itself is rendered in red with no lighting so that all pixels are either red or black. The number of red pixels in the column of a particular x value are added up and inserted into the data array element corresponding to that x value. The maximum value of an element in the array is therefore the y-resolution of the rendered image – this occurring when an entire column is rendered red.

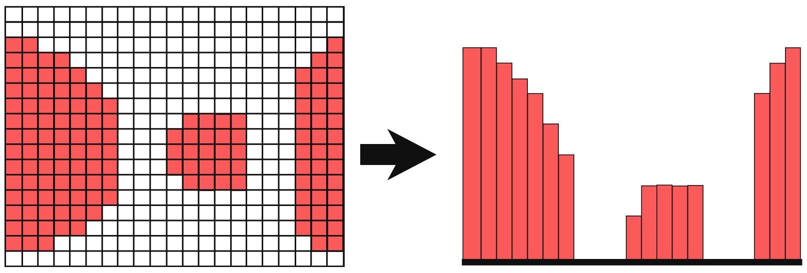


Figure 6.7: *The conversion of the world rendered from the point of view of a particular agent into the data array provided to the agent's brain.*

In order to allow an agent to distinguish between different agent types and world geometry, two different types of senses are created: one for seeing world geometry and another for sensing other agents. Agents in the world are organised into groups and a vision sense allows a particular group to be specified. Special cases such as seeing all other agents, seeing only agents in an agent's own group and seeing agents in all foreign groups are also allowed. An agent is not limited in the number of senses it can have and so may have as many "agent vision x" senses as required.

6.4.2 Synthetic Hearing

A simpler hearing sense, based on that used by Massive, is also reproduced. The method of converting a sense's data array into a fuzzy variable's value using a membership function is standard across all the senses. The difference between one sense and another is the manner in which the data array is generated.

For hearing, the data array is of length 360 and represents the angles, in degrees, from a sound comes. The first element of the array represents sound coming from $-180°$, with the last element of the array representing the sound coming from $179°$. The values of the array elements are either zero (for no sound) or one (for sound heard).

Distance from the sound's source is modelled not so much as how loud a sound is, but rather how noticeable. Each sound produces a solid bar in an agent's sense (as shown in figure 6.8). This bar is centred on the source of the sound and its thickness is determined by the distance of the source from the agent. Parameters specifying the maximum distance that a sound may cover and maximum thickness of a sound bar allow control of the sensitivity and accuracy of an agent's hearing. The use of these bars enables an agent to be influenced more by sounds that are nearby than those that are in the distance. We found that this sense worked well, although there are other possible methods to achieve a similar effect.

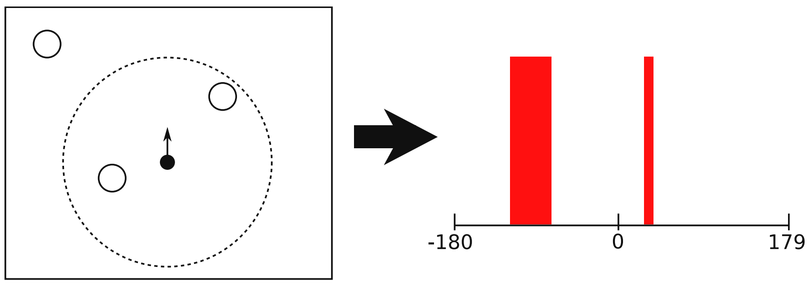


Figure 6.8: *The sound sense maps sound sources to bars in the sense data array. The width of the bar is determined by the distance from the agent. The dotted line shows the range of the agent's hearing sense.*

The hearing of an agent is based on the angle between the agent and the sound source. An angle measures a rotation about an axis and so we must define both this axis and what is considered zero rotation about this axis. Since an agent senses from its own frame of reference, we measure rotations around the vectors describing an agent's orientation. Agents three orientation vectors (show in figure 6.2): *look*, *up* and *right*. We create one hearing sense for each of these vectors - each sense measuring the rotation required around a particular orientation vector in order to best align it with the sound source. We describe the manner in which the sense which measures rotation around the *right* vector is implemented as the other two function similarly.

As the three orientation vectors are orthogonal, a rotation around the *right* vector takes place in the plane defined by the *look* and *up* vectors. The first step in calculating this angle is to project the position of the sound source onto this plane. We denote the *look*, *up* and *right* vectors of the agent as l, u and r

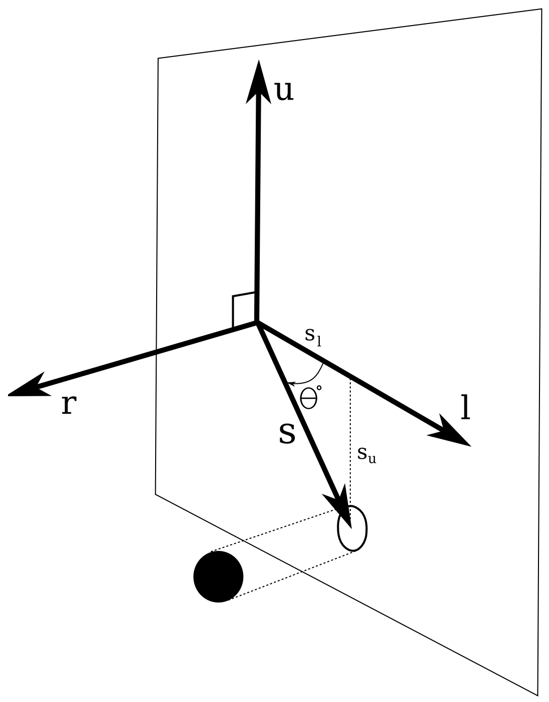


Figure 6.9: *The angle of a sound measured around the right (r) orientation vector. The position of the sound source (shown as a black sphere) is projected onto the u-l plane and then the required rotation determined.*

respectively. The vector s is in the direction of the sound source. We calculate the values

$$s_l = s \cdot l$$

and

$$s_u = s \cdot u.$$

as shown in figure 6.9. The angle of the rotation about r, θ is then found by taking the arctangent of the ratio of these coordinates as follows:

$$\theta = arctan(\frac{-s_u}{s_l}).$$

Here θ is the angle measured from the *look* vector, away from the *up* vector, hence the use of the negative s_u value. Special care must be taken in the case where s_l is zero. In this case, the sign of s_u determines whether the angle is 90°, for $s_u < 0$, or 270°, for $s_u > 0$.

Once the angle of the sound source in the particular plane has been determined, it only remains to apply this information to the sense's data array. A sound source generates a bar of data in this array, based on its distance from the agent. This distance, $||s||$, is the magnitude of the difference in position between the agent and the sound source, projected onto the relevant plane. The bar is centred on the index of the data array that represents the closest degree to the calculated angle. The width of the bar is determined by two parameters that give the range of the sense, c_{range} and the maximum width of the bar, c_{bar}. The width is then calculated as

$$c_{bar}(1 - \frac{||s||}{c_{range}}).$$

6.4.3 Target Sense

There are a number of senses implemented that provide more exact information to the agent. These senses are calculated in a similar manner to the hearing senses described above. The difference is the manner in which the sense information is represented in the sense data array. With the hearing senses each sound source produced a bar of values but in these more exact senses, only a single array element is given. This change results in greater accuracy from the sense but does not provide the agent with any distance information. These senses are created to provide information to the agent which does not require such information.

The simplest of these senses are the *target up* and *target right* senses. These senses allow the user to provide a target point to the agent which may then be used for navigation. These senses provide the agent with the rotation angles required to orientate the agent toward the target. The "up" and "right" designations are for the vector about which the rotation is calculated. Like the hearing sense, the data array represents the angles around a particular orientation vector ranging from -180 to 179. The angle calculation is also identical to that described above for the hearing sense. Once this angle is found, the element of the data array that corresponds to this angle is set to 1 with all other elements set to 0 as shown in figure 6.10. The use of only two senses while there are three orientation vectors is simply because there was no need for a third sense in our experiments. Using only the two angles an agent is able to orientate itself towards the target point. The target point is set by the parameters provided to the sense when it is created.

The fuzzification of these exact senses is performed by the Fuzz-Max nodes. If a standard Fuzz node is used, as would be the case for the vision or hearing senses, the averaging effect of the fuzzification almost ignores the single value provided by these senses. The Fuzz-Max node, in contrast, is sensitive to the maximum value in the data array and so provides the brain with a usable value.

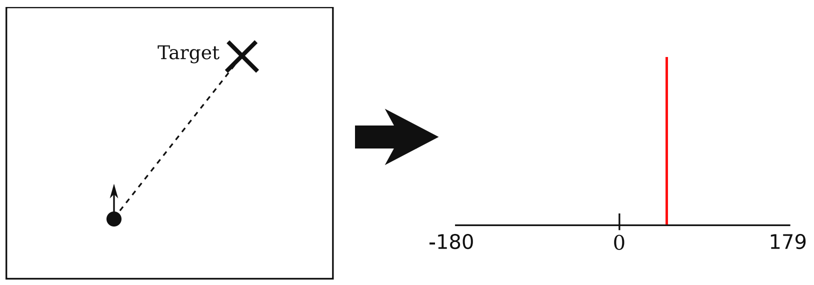


Figure 6.10: *The target sense produces a single value in the data array based on the rotation required to orientate the agent toward the target. The target sense shown determines this rotation around the "up" axis.*

6.4.4 Group Senses

When creating flocking behaviour, similar to that of Reynold's birds (see section 2.2.2), it is useful for an agent to have information on its flock mates. In order to meet this need, a number of group senses were created. Similar to the target senses are a pair of *group centre* senses. These two senses function identically to the target senses except that the target point is the centre of the particular agent's group. Calculating this point is a simple matter of averaging the positions of all the agents in the group.

The *group velocity* senses allow an agent to be aware of the average velocity of the agents in its particular agent group. Although the idea behind these senses differs somewhat from the hearing, target and group centre sense, the implementation is almost identical. If one compares *group velocity up* sense to the *target up* sense, the only difference is that the rotation is calculated to align the agent's look vector with the group's velocity. As with the previous two sense types, only two rotations are needed as this is all that an agent requires in order to match the velocity of the other group members.

There are a number of senses in which the information that the sense provides is a single real number. An example of this is the *Group Distance Sense*. Since neither of the two other group senses provide information on the distance of the group centre from the agent, an additional sense is required to provide this information. The distance of an agent from its group is determined by finding the centre of the group and then calculating the magnitude of the difference between that position and the agent's own position. One difficulty that must be overcome is the manner in which this one dimensional data is provided to the brain by making use of a two dimensional data structure.

The distance sense provides distance information in the form of a ratio. The maximum range of the sense is required as a parameter at the sense's creation and, assuming that the distance to the group is less than this maximum, the information given is the ratio of this distance divided by the maximum. The data array therefore takes on values between zero and one with all the values in the array given the same value. Due to the design of the fuzzification process conducted by the Fuzz node, any membership function used will produce this same value from the 2D data array. A data array of length one is used for efficiency and the agent is able to make use of this distance information in its inference process.

6.4.5 Constant Sense

In the course of creating crowds for testing the constraint-based control, it became useful to have a constant value available to the agent. This is provided in the form of the *Constant Sense*. This sense takes a single parameter between zero and one and provides it to the agent brain in a similar manner to the *group distance* sense described above.

6.4.6 Noise Sense

In order to allow variation in behaviour it is useful to provide random noise. This sense is used, for example, to produce the erratic movement of the mouse agents described in chapter 8. The noise sense provides a value between zero and one to the brain, taking a single parameter that determines the speed at which this value is varied. It is, however, important that the same noise is produced by each run of the simulation, otherwise different behaviour may occur in two identical crowd simulations. This requirement is met by giving each sense object a dedicated random number generator that is seeded with the index of the sense in the array of sense objects. In this way each noise sense is seeded with its own unique number but the same number is used on each simulation run.

6.4.7 Control Sense

The *control* sense provides an agent with a degree of self-awareness and allows the behavioural rules to take into account the agent's own state. A control sense object is associated with a single control and, since each control is a single valued variable, provides a single value to the brain. The manner in which this is done varies from the other single valued senses since the control variable takes on discrete values based on its particular control step size. The data array provided by the control sense has the same number of elements as a control has possible values. The values of the data are all set to zero except for the particular element that corresponds to the current value of the control variable. This element is set to one.

By making us of a Fuzz-Max node, the brain is able to access the value of this variable. The behaviour of this fuzzification is, in fact, closest to the theory involving membership curves presented in section 4. The single value of the control variable may be fuzzified by assigning the input node the value that it produces from the membership function.

6.4.8 Additional Senses

The flexibility of the sense interface allows additional senses to be created as needed. This flexibility is demonstrated in the work of Perkins *et al.* [PJM08] where our crowd simulation system was adapted to make use of a framework for providing spacial awareness to agents. The information from the framework is made available to the agents through additional senses that query the spatial data structure. The agents are therefore able to make use of this additional information and so act more intelligently.

The use of different sense models is key to the realistic animation produced by systems such as Massive. The implementation of the 2D data array as a means for providing sense information is flexible enough to cater for the various senses that Massive makes use of as well as others that are needed. Most importantly, this framework allows the implementation of a sense of vision and hearing, similar to those provided by Massive.

6.5 Agent Initialisation and Definition

The system described in previous sections simulates a crowd made up of autonomous agent objects moving in a virtual world, governed by a global world object. We have yet to cover the manner in which the world and agents in it are defined, stored and initialised. Since the system is intended for research purposes, a high degree of usability is not required. However, to facilitate testing using an expert user, some form of authoring tool is necessary.

World Object
The world is initialised with a world file which contains the information and file names required to define a crowd simulation. The environment is given in a geometry file which contains an XML description of the 2D polygons that are used to define the world's geometry. The manner in which this is done is described in section 6.1.

Agent Groups
The agents are then specified in groups. Each group of agents has a group name, used in both the constraint satisfaction system and the agent's senses. The agents of a particular group all share the same body type, parameter vector and brain. Each agent has its own position and orientation specified in the world file.

Agent Bodies
The body type of an agent defines the state object which determines the control variables and geometry of an agent. Although some form of variation could be implemented at this level, we choose not to do so for simplicity's sake. The parameter vector specifies the senses that an agent has available, as well as any associated parameters for them such as vision resolution, etc. The parameter vector also specifies the control variables an agent makes use of, as well as the initial values of these variables.

Agent Brains
A group of agents have their brains defined by a brain file which is created and edited in the brain editing tool. A brain file contains a list of brain nodes along with their associated parameters. These parameters include the nodes upon which a particular node depends as well as the parameters of the membership function, in the case if input and output nodes, or other parameters such as scale factor. The nodes are ordered by dependency such that a node is defined before any nodes that depend on it.

We created a simple brain editing tool which displays each brain node along with its parameter values. In a similar manner to the Massive brain editor, the brain nodes are displayed in a graph with edges representing the dependencies of the nodes, as shown in figure 6.3. This tool allows the user to create and link in new nodes as well as edit existing ones. The brain information is saved in a flat text file that is human readable, though direct editing of this file would be unwieldy due to its length and lack of obvious structure.

The crowd simulation system described has been used to create a number of crowds and has proven flexible and powerful. The system is based on the principles of the Massive system which makes it a solid base for our work in constraint-based control.

6.6 Example Brain

In order to illustrate the working of our crowd simulation system, we provide an example based on the Car agent, which is described fully in chapter 8. Figure 6.12 shows part of the steering sub-system of the agent's brain, which attempts to prevent collisions with the world geometry. This behaviour has three input nodes, named "obstacle left", "obstacle right" and "obstacle centre". These three nodes represent linguistic values describing the position of any obstacles that an agent may be aware of and may be true to differing extents. The sense attached to each of these nodes is a world geometry vision-x sense (as described in section 6.4.1). The two output nodes take the combined values of the input and react accordingly – turning left if there are obstacle ahead and to the right and turning right in the opposite situation.

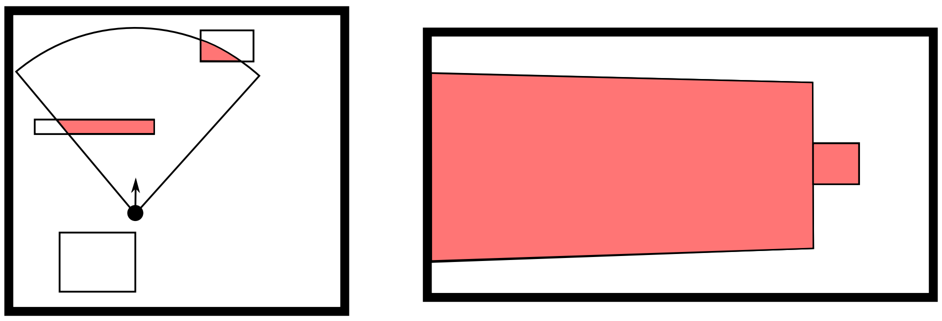


Figure 6.11: *Example vision input for a Car agent. The image on the left shows an overhead view of the agent with its vision arc – the visible obstacles are highlighted in red. The rendering of the scene from the agent's point of view is shown on the right.*

For particular sensory information, shown in figure 6.11, the values of the brain nodes are calculated as seen in figure 6.12. The brain input is fuzzified, combined and defuzzified (see figure 6.13) to give the control, *turn angle*, a value of 13°. As a positive value results in the agent turning to the right, this behaviour avoids the approaching wall in the environment.

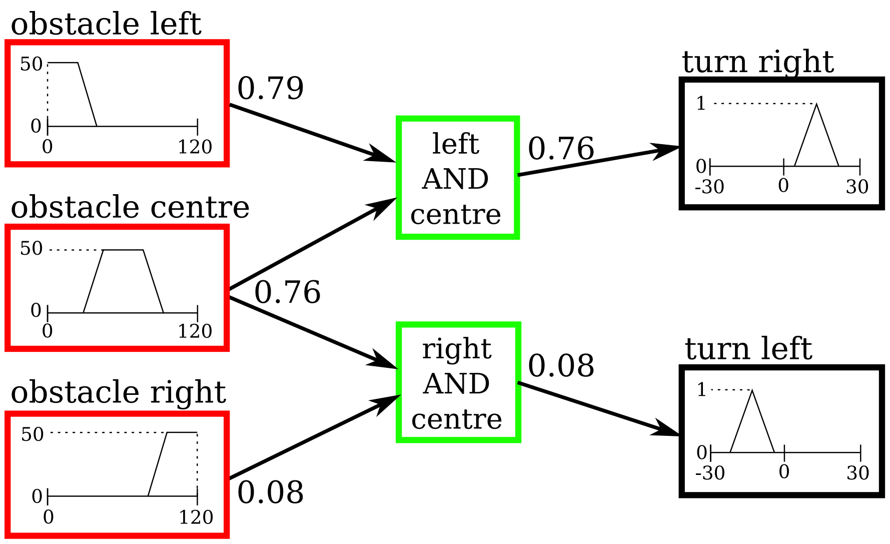


Figure 6.12: *A subset of the brain for the Car agent. This fragment has three input nodes (shown in red) each applying their membership curves to a world geometry vision-x sense. These nodes are combined by the two logical AND nodes (shown in green) and provided to the two output nodes (shown in black). These output nodes alter the agent's "turn angle" control according to their particular membership curves. The values given for each node are for the example input of figure 6.11*

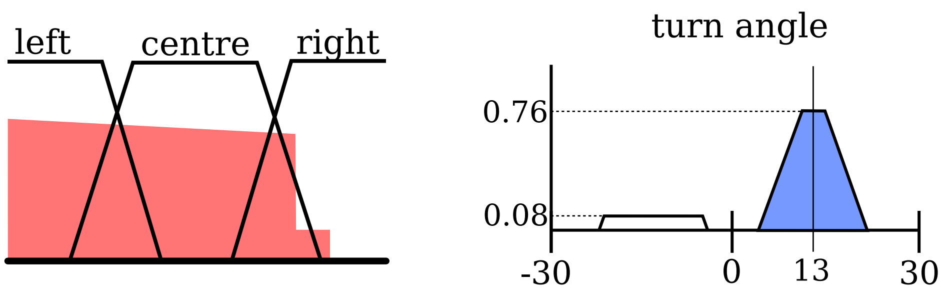


Figure 6.13: *The fuzzification and defuzzification steps undertaken by the agent brain. The input from figure 6.11 is fuzzified according to the membership curves shown on the left. The result of the fuzzy inference is combined and defuzzified as shown on the right to give a turn angle of 13 degrees.*

CHAPTER 7

Constraint Satisfaction Implementation

The control mechanism proposed in this dissertation is based on user-specified high-level constraints. The system implemented (shown in figure 7.1) aims to test the feasibility and possible usefulness of this form of control. As this is a first attempt at providing general control in this form, a number of simplifying implementation decisions were made. Our primary aim in creating this system is to provide a proof-of-concept for constraint-based control.

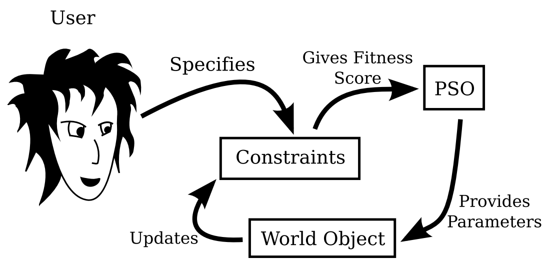


Figure 7.1: *The organisation of the full Constraint Satisfaction System.*

7.1 Constraint Specification

Our aim in providing constraint-based control is to allow the user to freely control a crowd's behaviour. In keeping with this aim we permit constraints to be as general as possible, allowing them to be specified over any part of a world object and over any time interval during a simulation's run. The general structure of a constraint is therefore intentionally vague. A constraint is specified as an object which is updated at each timestep of the simulation and has access to the entire world object. A constraint must provide a value which indicates the degree to which the constraint has been met by the simulation. Care must be taken in defining the behaviour of a constraint before a value is available. For example, a constraint might specify that a certain number of agents be alive at a given time. Before the simulation reaches this time, the value of the constraint is not well-defined. Generally, we assign a constraint the value of 0 if a value is not available. Making sure that a simulation runs long enough for all the constraints to be determined is up to the user, though this is not difficult to ensure.

The other limitation that we place upon constraint objects is that they must apply to only one agent group. This is imposed so the constraint satisfaction system (CSS) may determine which parameters it should alter in trying to meet the constraints (the parameters are discussed in section 7.2). Constraints could be extended so that are allowed to apply to any number of groups without difficulty though this was not required by any of our test cases.

Each constraint is also required to have a weight. This weighting is based on how important it is that this constraint be met. A user may quite possibly specify constraints that are incompatible – both of them cannot be met at the same time. The system therefore requires a means of deciding which constraint is more important. In practise we found that this did not occur as frequently as we initially expected. A further discussion on the use of these weights is included in the section on optimisation (section 7.4).

We implement constraint objects in a hierarchical, object-oriented manner so that constraints are specified as subclasses of the base constraint object. Additional constraint classes were created, inheriting from the super constraint class, for each of the different forms of control that we required. As explained in the section on system design (section 5.3), the focus of this research is not the manner in which constraints are used, but rather the possibility of them being used for control purposes. The full list of constraint that we implemented, along with their intended use, is as follows:

The Agent Collision Constraint is used to reduce the collisions between two groups of agents. This constraint is useful in situations where the agents are behaving correctly but there are some collisions between them. The tweaking that is required in order to remove the collisions is therefore done automatically. This is a useful constraint to have in place when specifying other constraints as any changes made by the system should not cause additional agent collisions.

The relevant parameters of this constraint are the names of the groups of agents (these may be the same in order to stop collisions between agents in the same group) as well as minimum distance to be kept between two agents. A bounding sphere around each agent is used to determine collisions. The bounding sphere with a radius of half the minimum distance is used. The value of this constraint is given as the number of agents in the first group who have not been involved in a collision, divided by the total number of agents in the first group. This focus on the first agent group only is due to the default behaviour of our crowd simulation which removes dead agents from the simulation. When a collision between two agents occurs, both agent's state is set to dead and they are removed.

It is important to note that the manner in which this value is calculated is arbitrary from the point of view of the CSS. As long as the constraint's value is consistent with the degree to which it is met, the CSS should be able to use it effectively. The value of the constraint decreases with every collision, as required, and is in the range $[0, 1]$. A different version of this constraint could easily be created which calculates the constraint value in a different way. Further discussion on future research into constraints is provided in section 9.2.

The Geometry Collision Constraint is similar to the Agent Collision constraint except that it checks for collisions between an agent group and the world geometry. We initially created this constraint in order to stop the system from driving agents through walls in order to avoid collisions with other agents. It has also proved useful in tweaking behaviour that is mostly correct as a default addition in order to avoid changes that cause agent-environment collisions to occur. The parameters consist of the name of the agent group as well as a minimum distance to be kept between an agent and a wall. The agent is given a bounding sphere of this radius and any intersection of this sphere with the world geometry is considered to be a collision. The value of the constraint is calculated as the number of agents in the group that have not yet collided with a wall, divided by the total number of agents in the group.

The Agent Control Constraint is a constraint placed on the control variables of groups of agents over a set time frame. It requires the name of an agent group, the name of a control variable, a target value for this control variable and a time interval. The value of the constraint starts at 1 and a fraction is subtracted each time step, for each agent, based on the difference between the target value and each agent's actual control value. This constraint is unlikely to ever be completely met by a simulation as even a small variation in the control variable results in the constraint value dropping below 1. Rather, the aim of this constraint is to provide a means of ensuring that agents' controls are kept close to the optimal value. The initial use of this constraint was to stop the CSS from altering the agents behaviour such that they stop moving in order to avoid collisions. By applying this constraint to one of the movement related controls, the agents were constrained to keep moving. In practise we found the line-crossing constraint, described below, to be more useful for this purpose as that constraint can more easily be met completely.

The Line Crossed Constraint is actually implemented as four different constraints which constrain a group of agents to have crossed a specified line by a certain time. Since all four are almost identical save for the orientation of the line and the direction of the line crossing, we describe only one of them.

In all cases the line lies on the "ground plane" of the environment where $z = 0$. The one constraint we take as a representative defines a line $y = a$ for some parameter a. The agents in the agent group to which the constraint applies are all required to be on the one side of the line such that $p_y > a$ where p_y is the y component of the particular agent's position vector. The final parameter of the constraint is the time by which this must take place.

The constraint's value is initially set to 0 and stays this way until the elapsed time of the simulation is greater than the time parameter provided to the constraint. At this point the value of the parameter is calculated as the number of agents in the agent group that are over the line, divided by the total number of agents in the agent group.

Although these constraints could be created for arbitrary lines, we found that our environments were usually aligned to a particular axis and so it was simpler to define lines parallel to the world x and y axes. These parameters were initially created for racing scenarios where we wanted to ensure that agents continued moving in the correct direction around the track. Examples of this constraint being used in this way occur in the *Car Bridge Crossing* and *Car Cornering* scenarios (see section 8.2).

The Group Crossed First Constraint is similar to the previous constraint in that it was first used in a racing car scenario. The control required is that a particular agent or group of agents be the first to cross a particular line in the world. The lines are defined in a manner similar to the previous constraint.

There is no time parameter for this constraint. Rather, the constraint keeps track of the number of agents that cross the line before all the agent in the specified agent group have done so. The constraint value is given by one minus an "undesirability" term calculated as the number of agents who cross the line ahead of the specified group, divided by the total number of agents in the simulation.

The Number Alive Constraint provides exact control over the number of agents in a particular group that are "alive". This state is usually changed by collision or, in the case of the War Robot, being shot by another agent. The constraint has no time frame and so provides useful information at all stages of the simulation. The parameters required are simply the agent group's name and the number of agents that should be left alive. At the constraint's initialisation, usually the beginning of a simulation, the difference between the number of agents alive initially and the required number of agents alive is recorded as *init difference*. At any particular time step, the value of the constraint, v, is calculated as

$$v = 1 - \frac{|\textit{current difference}|}{\textit{init difference}}.$$

This constraint is designed to be placed on simulations where a number of agents are required to die, such as a battle scenario. A different constraint would have to be constructed if the requirement is to have all the agents in a particular group survive as this would result in a division by zero error in our particular implementation. This calculation also weights the difference in number of agents alive from the target number evenly, regardless of whether there are too many or too few. Both cases are considered equally undesirable. The calculation of the constraint value may result in a value of less than 0. This is because *init difference* might be less than the target number of agents alive. Since this is taken as the worst case, a very small number of agents left alive might result in $|\textit{current difference}|$ being greater than *init difference*. In this case loss of information is unavoidable and the constraint value is clamped to 0.

The constraints described above are the constraints that were implemented for the testing of the CSS as detailed in chapter 8. They serve as examples of constraints that could be created in order to control crowds, not a complete list. The aim of this research is to test the possibilities of this form of control. Future work may determine a useful list of standard constraints to be implemented.

7.2 Simulation Parameters

The CSS attempts to meet the constraints specified by the user by altering the behaviour of a group of agents. This is done by altering the parameters of the brain that define the membership functions of the different brain nodes. These brain nodes are used as linguistic values defined by some form of fuzzy set. By altering the membership functions of the brain nodes, the CSS changes the agent's understanding of the meaning of the particular linguistic value. For example, an output node with the name "hard left turn" makes use of a fuzzy set over the universe of possible turn control values. By altering the membership function of the output node, the CSS may change what an agent understands a hard left turn to be.

The brain parameters that the CSS requires are determined by the agent groups that the constraints are applied to. This decision is a limitation since the global behaviour of the crowd may well depend on all the agents in the simulation, not just those that the constraint is applied to. Despite this, it is desirable to reduce the number of parameters that may be altered as additional parameters increases the dimension of the search space. We therefore make the choice to only alter the brain parameters of groups for which the constraints are specified as altering these agents is likely to have the largest effect on the value of the constraints.

Once the list of agent groups is known, the world object is queried for the brain parameters of each of these groups. Since agent groups share the same brain, only the set of brain parameters from one agent in the group is needed and is supplied by the world. These parameters are organised in a simple parameter array to which each brain node appends its relevant parameters in turn. Brain nodes that do not make use of membership functions do not have any parameters to add. The exception to this are the Scale and Power nodes. These nodes are based on the concept of hedges which themselves alter the meaning of a particular linguistic value by changing its membership curve. Since the effects of these nodes are in keeping with our aim of altering an agents understanding of the linguistic values with which it reasons, these two parameters: scale and power, are also included as brain parameters to be altered.

The membership functions implemented by our crowd simulation system have either two, three or four parameters depending on their type as seen in figure 4.2. These functions are defined over different domains depending on the sense or control variable that they are applied to. In order to create a more uniform search space, the parameters that define a membership function are normalised to lie between zero and one. When the parameters are returned, they are mapped back to their original domain.

Since these normalised membership functions are defined over a domain of $[0, 1]$, the parameters returned by an agent's brain are themselves constrained: none of the parameters of a membership function may lie outside of the function's domain. There is also a specific ordering of parameters that must be maintained. With a triangle function, for example, the parameter that defines the peak of the triangle must lie between the other two parameters. A careful distinction must be made between *crowd constraints* – the constraints a user specifies in order to apply control to a crowd – and these *parameter constraints* – constraints on the values that the brain parameters may take.

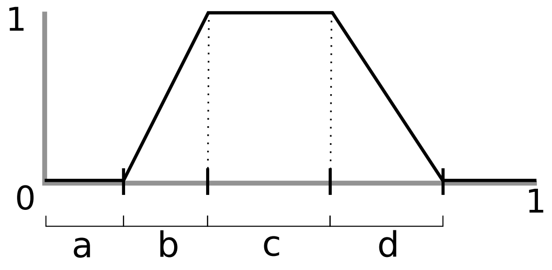


Figure 7.2: *The encoding of the parameters of a membership function as differences between the curve's parameter values.*

In order to reduce the number of parameter constraints, we encode the parameters as a set of differences as shown in figure 7.2. Each parameter is encoded as the difference between itself and the parameter less that it with the smallest, left-most parameter provided outright. The ordering and domain constraints are thus reduced to two constraints per membership function. For a normalised function with parameters $x_1 ... x_n$ where $n \in \{2, 3, 4\}$, the encoded parameters $e_1 ... e_n$ are

$$e_1 = x_1$$
$$e_i = x_i - x_{i-1} \text{ where } i \in [2..n].$$

The two constraints for these parameters are

$$e_1 \geq 0$$

and

$$\sum_{i=1}^{n} e_i \leq 1$$

These parameters constraints are implemented as *p-constraint* objects (as opposed to the crowd level *constraint* objects) which are created by each brain node as its parameters are requested by the world. The p-constraints record the constraints as shown above as well as the indices of the parameters in the parameter array that is being populated. These p-constraints are added to a separate list that is simultaneously created. Once the world has collected parameters and p-constraints from each of the required agent groups, the two arrays are supplied to the CSS.

The action of the CSS is to alter the parameters in the parameter array while maintaining the constraints given by the p-constraints array. When running a test simulation, an altered parameter array is supplied to the world object which is responsible its distribution among the different agent objects. The world does this by reversing the order of the parameters and then allowing each brain node to take the number of parameters that it added when the parameter array was created.

7.3 The Search Space

The task of the Constraint Satisfaction System (CSS) is to find a set of parameters that satisfy the parameter constraints of the agent brains, create crowd behaviour that satisfies the crowd constraints supplied by a user and change the behaviour of a crowd as little as possible. This problem is an example of a Constraint Satisfaction Problem in which it may not be possible to satisfy all the constraints.

As the crowd-constraints are real valued functions, the problem may be stated as a form of multi-objective constrained optimisation problem as follows. Each of the crowd-constraints, c_i, is an objective function that needs to be optimised. These functions take a parameter vector, p, that satisfies the parameter constraints of the system and maps it to a value between zero and one. The constraint-functions are evaluated by running a simulation with the particular parameter vector. The n parameters supplied by the world object form an n-dimensional search space over which the objective functions, or crowd constraints, are defined. The problem is, therefore, to find p such that the objective functions, c_i, are maximised.

Before deciding on a method to perform this optimisation, properties of the search space and objective functions must be considered.

The first property that is of importance is the *continuity* of the objective functions and whether their *derivatives* are available. When one considers the arbitrary constraints described in section 7.1, it is clear that requiring continuity and differentiability would place considerable limitations on the constraints that might be specified. In fact, when the local sensory model of agents, the complexity of their brain and the emergent behaviour that results from the interactions of agents is taken into account, it becomes clear that differentiability or continuity are not realistic requirements for useful constraints.

A second property of the objective functions that is of great importance to the optimisation problem is the *cost* of evaluating an objective function. Even though all the constraints may be evaluated simultaneously, this still requires running a crowd simulation. Although these simulation runs do not require any visualisation, the simulation logic must be run in full. To do otherwise would be to produce constraint values that would not match up with those given by a true simulation. The running time of a full crowd simulation, even without visualisation, is considerable (at least a few seconds).

An additional property of the constraints is their lack of *description*. These constraints are general and can not be formalised or specified before the user actually creates them. Effectively, these objective functions are black boxes which bear no analysis as their form is not known at design time.

Turning to a consideration of the search space, two properties are of interest. The first is the *dimensionality* of the space. Each of the parameters may be varied independently and, since there are up to four parameters per input or output node and one per scale or power node, the number of parameters can become quite large. In some of the test scenarios, involving only two different types of fairly simple agents, there were over one hundred parameter giving a similar number of dimensions to the search space. Various methods of reducing the dimensionality of the parameter space might be considered, but this is beyond the scope of our work (see section 9.2).

The parameters, however, cannot vary freely as they are under the domain constraints specified by the p-constraint objects. In this way the search space is constrained. A useful property of the space is that, although it is constrained, the feasible region in which potential solutions may be found is convex. Unless specifically noted, reference to the search space implies the feasible region of the search space.

The definition of convexity requires that any two points in the space may be connected by a line segment that does not leave the feasible region. This is equivalent to the statement that for two points $\mathbf{x}$ and $\mathbf{y}$ in the space, the point $\mathbf{z} = (1 - \lambda)\,\mathbf{x} + \lambda\mathbf{y}$ with $\lambda \in [0, 1]$ is also in this feasible region.

We show that the search space is convex by considering two points $\mathbf{x}$ and $\mathbf{y}$ in the feasible region and demonstrating that $\mathbf{z}$ is indeed feasible. In order for these two points to be in the space, they must satisfy a number of constraints of two different forms. The upper limit domain constraints are of the form

$$\sum_{i \in I} x_i < 1$$

where I is the set of indices for a particular membership function's parameters. The lower limit domain constraints are more simply of the form

$$x_i > 0$$

where i is the index of the first parameter for some membership function. Our approach is to show that if $\mathbf{x}$ and $\mathbf{y}$ satisfy a particular constraint, then so does a point that lies on the line segment between them, $\mathbf{z}$.

We first deal with the upper limit constraints. Given $\mathbf{x}, \mathbf{y} \in \mathbf{R}^n$ such that $\sum_{i \in I} x_i < a$ and $\sum_{i \in I} y_i < a$ for some $I \subseteq \{1..n\}$ and $a \in \mathbf{R}$.

Take $\mathbf{z} = (1 - \lambda)\,\mathbf{x} + \lambda \mathbf{y}$ with $\lambda \in [0, 1]$. Then

$$\begin{aligned} \sum_{i \in I} z_i &= \sum_{i \in I} [(1 - \lambda)\, x_i + \lambda y_i] \\ &= (1 - \lambda) \sum_{i \in I} x_i + \lambda \sum_{i \in I} y_i \end{aligned}$$

Without loss of generality, assume $\sum_{i \in I} x_i \geq \sum_{i \in I} y_i$. Then,

$$\begin{aligned} (1 - \lambda) \sum_{i \in I} x_i + \lambda \sum_{i \in I} y_i &\leq (1 - \lambda) \sum_{i \in I} x_i + \lambda \sum_{i \in I} x_i \\ &= \sum_{i \in I} x_i \\ &< 1 \end{aligned}$$

And so $\sum_{i \in I} z_i < 1$

We then consider the lower limit constraints. Given $\mathbf{x}, \mathbf{y} \in \mathbf{R}^n$ such that $x_i > 0$ and $y_i > 0$ for $i \in \{1..n\}$.

Take $\mathbf{z} = (1 - \lambda)\,\mathbf{x} + \lambda \mathbf{y}$ with $\lambda \in [0, 1]$. Then,

$$z_i = (1-\lambda)\, x_i + \lambda y_i$$

Without loss of generality assume $\sum_{i \in I} y_i \leq \sum_{i \in I} x_i$. Then,

$$\begin{aligned} (1-\lambda)\, x_i + \lambda y_i &\geq (1-\lambda)\, y_i + \lambda y_i \\ &= y_i \\ &> 0 \end{aligned}$$

And so $z_i > 0$. This shows that if two points are in the space, all the points between them are also in the space and, hence, the search space is convex.

In summary, we have a multi-objective constrained optimisation problem with discontinuous objective functions that are expensive to evaluate. The optimisation is over a convex space of high dimension. These properties are taken into account in our choice of optimisation technique.

7.4 Optimisation Techniques

As described in the previous section, satisfying the user-specified constraints is a multi-objective optimisation problem as there is no guarantee that the set of constraints specified by user can all be completely met. In this case it is necessary for some of the constraints to be relaxed. There may, therefore, exist a set of solutions which are, in a sense, all equally good. This *Pareto-optimal* set is loosely defined as the set of solutions such that each of the solutions cannot produce an improvement in any of the objectives without worsening one of the other objectives.

7.4.1 Compromise Strategy

The decision of which solution out of this set should be chosen requires some form of compromise strategy. These strategies may be applied *before* the optimisation process by specifying some preference among the conflicting objectives. Alternatively, the preference may be specified *after* the optimisation process by selecting a solution from the set of generated possibilities. The third option is to have the preference specified *during* the optimisation procedure, basing the compromise on information received from the process.

The expense of evaluating the objective functions is a major factor in determining our approach. Since a post-optimisation decision strategy requires the generation of a number of pareto-optimal solutions, it is likely that such an approach will require more function evaluations than deciding how to compromise between objectives before the optimisation begins. It is also worth noting that much of the discussion and work on multi-objective optimisation is applied to design or control processes which have a number of objectives that are likely conflicting. This is not necessarily true in our case.

The constraints placed on a crowd are for the purposes of controlling it. In visual effects specifically, this is done in order to create global crowd behaviour that is believable or plausible. It is reasonable to assume that the control will be applied by a user with this goal. Since the aim of the specified constraints is to create a crowd that meets the behaviour expected of a crowd which exists in the user's imagination, the constraints work toward a single goal. The constraints specified are unlikely to be contradictory conceptually as the crowd that the user envisions is able to meet all the constraints perfectly. The way in which the constraints may conflict is based only on the limitation of the simulated crowd's model – made up of sense specification, brain structure, inference method etc. Because of this conceptual cohesion, we may expect to find objective functions that are less difficult to balance than the conflicting objectives of other optimisation problems – cost and performance trade-off that a motor vehicle designer faces, for example.

These considerations prompt the use of a simple compromise strategy that allows for a minimal number of evaluations of the objective functions. It is also vastly more efficient to evaluate all the objective functions at once as they may all be computed by running a single simulation. This is not always the case for optimisation techniques that produce a pareto-optimal set. We choose to use the simple weighting method [LYW03] in order to compromise between objective functions.

This method creates a single objective function by taking a linear combination of the multiple objectives. Each objective is assigned a scalar valued weight. Since the search space is normalised, this weight may be considered to represent the user's preference for different objectives and is the reason for the weight parameter of each constraint object (see section 7.1). For the individual objective functions $f_1, f_2, ..., f_n$ assigned the weights $w_1, w_2, ..., w_n$ respectively, a single objective function, f, is defined as

$$f(\mathbf{x}) = \sum_{i=1}^{n} w_i f_i(\mathbf{x})$$

for $\mathbf{x}$, an element of the parameter space.

It is important to recognise the properties of this weighting method, some of which are desirable and some of which may be problematic. This weighting scheme is best used if the objective space is convex. If this is not the case then parts of the objective space may be unreachable and so some solutions may not be found [LYW03]. Since our objective functions are arbitrary, determined when the user defines the controlling constraints, there seems to be no easy way to guarantee this. We have shown that the feasible region of the search space is convex. This may, however, not be true of the objective space.

An advantage of the simple weighting compromise method is that it reduces a multi-objective optimisation problem to a single objective problem. We may then make use of the wealth of single objective optimisation methods available in order to find solutions. An obvious disadvantage is that a user must indicate his preference for the different objectives in advance – before the possible trade-offs are known. Since the objectives are intuitively understandable constraints set by the user, we do not expect this to limit the usefulness of this control method. Although we have presented our motivation for choosing the simple weighting compromise method, a proper comparison of different strategies would require extensive implementation and experimentation and is beyond the scope of this dissertation (see section 9.2).

7.4.2 Global Optimisation Options

Given a single, combined, objective function, it remains to choose a method for optimisation. There are a host of options available for single objective optimisation. However, due to the properties of the search space and objective function, the possible choices are severely limited.

Optimisation techniques can be broadly classified under either *calculus-based* techniques or *direct-search* methods [LYW03]. Calculus-based techniques make use of properties and analysis of the objective function in order to suggest and eliminate possible solutions. Included in these techniques are wealth of mathematical optimisation techniques such as non-linear programming. Unfortunately, the black-box nature of our objective function makes the application of these techniques difficult however. The function cannot be expressed analytically and the gradient is not available. There are methods to approximate the gradient of the objective function but these require a number of function evaluations and may still be unreliable, especially in the presence of the expected discontinuities.

Of the direct-search methods, we are interested in those that are applicable to *global* optimisation. The major challenge of global optimisation methods is the avoidance of local maxima (we assume that the objective function is to be maximised). The manner in which global optimisation methods attempt to achieve this goal may be broadly divided into two categories: *deterministic* and *stochastic* methods.

The deterministic methods often apply some form of heuristic to avoid or escape from local maxima. A characteristic of these methods is that they, at least implicitly, explore the entire search space. This requires either some additional knowledge of the objective function or a very large number of function evaluation, neither of which are possible for our problem. Stochastic methods make use of some form of probabilistic judgement to avoid the local maxima [PV02a].

From the discussion above, we have ruled out a large number of optimisation techniques. All the forms of mathematical programming and gradient-descent techniques are unavailable due to the black-box nature of our function and the remaining choices are limited by the cost of evaluating our objective function. Recently, a large amount of research has been conducted into a group of optimisation techniques called *meta-heuristics*. These optimisation techniques are designed to be used on a wide range of functions and are, therefore, well suited to the unknown nature of our set of crowd constraints.

A number of well known techniques fall into this grouping. Among them are *simulated annealing*, *tabu search*, *multi-start with clustering* and others. These techniques all make use of the evaluation of the objective function exclusively, requiring no gradient information, and so suit our problem well. The disadvantage of many of these techniques is the high number of function evaluations required [WSW04].

There are two possible approaches in dealing with expensive objective functions in the optimisation literature. The first approach is that of *meta-modelling*. The general aim of this approach is to substitute a statistical model for the objective function. This substitute is far less computationally costly to run than the true function and so may be used in the search in place of it. Samples of the objective function are used to either improve the statistical model by providing it with additional information on the objective function at areas where the model's estimated error is high, or to explore possible optimal points.

The disadvantage of such an approach is that the search is then based on the structure of the substitute. This makes implicit assumptions about the objective function and choice of underlying structure for the substitute may be problem-dependent [WSW04]. All optimisation techniques, however, assume some form of global smoothness or local correspondence between points. Such an approach may therefore be worth further consideration (see future work in section 9.2).

Another way in which expensive objective functions may be handled is through a class of stochastic optimisation methods that may also be categorised as meta-heuristics. This group of techniques are collectively known as Evolutionary Computation (EC) [RBP98]. The techniques in this group generally make use of a population of different prospective solutions that are then updated in cooperation to find the global maximum. The field has received much attention of late and popular EC methods include Genetic Algorithms, Evolution Strategies and Genetic Programming.

The advantage of these EC methods is not the number of function evaluations that they require, which is still quite high, but rather the parallel nature of their population-based approach. This ties in well with the trend of parallelisation as a way to deal with computationally expensive problems. As each potential solution is evaluated independently, more than one member of the population may be evaluated simultaneously if there are additional computing nodes available. Since the interaction between the members of the population is fairly limited (though this differs from method to method) the speed up of this search method is likely to be linear with respect to additional parallel processors. This offers a significant advantage over other optimisation methods and perhaps a means of dealing with the expense of running multiple crowd simulations. The use of parallel optimisation methods is particularly relevant in the visual effects industry where render farms (clusters of computing nodes used for rendering) are common.

The general algorithmic sequence for an evolutionary algorithm is the initialisation of a population of potential solutions, followed by the iterative improvement of these solutions until some terminating condition is met. The main improvement loop involves:

- *Evaluation* of the fitness of each member of the population.
- *Selection* of a sub-population based on their individual fitness.
- *Recombination* of the sub-population to form a new population.
- *Mutation* of some individuals in the population.

Two of the most widely described and best known forms of EC are Genetic Algorithms (GAs) and Evolution Strategies (ES). Of the two, EC is probably the most readily applied to our problem of real-valued optimisation due to the representation of individuals as real numbers. In GAs, individuals in the population are usually represented by a string of symbols that define the individual. The use of a GA for a real-valued problem introduces the considerable design element of selecting a suitable encoding. When making use of an EC approach, however, the parameter vectors taken from our agent brains may be used directly.

A standard ES scheme applied to optimisation makes use of the objective function as the measure of fitness. Since individuals are represented by real vectors, an individual's representation may be given directly to the objective function in order to produce a fitness score. Each individual, represented by its position in the search space, is also given a mutation vector of the same dimension. This vector is used to randomly mutate new individuals by simply adding it, randomly scaled, to the position of an individual. This mutation vector itself is then mutated. This mutation operation is key to the success of ES as the mutation of the individual is evolved in the same way as the individual itself, resulting in a capacity for self-adaptation.

The recombination operation, in contrast to that of a GA, is of lesser importance to mutation. With a GA, the string-encoding of the individual makes recombination of parts of an individual a meaningful process as worthwhile parts of a solution may be given to the next generation. This is less true of real-space as there is not a similar manner in which to decompose a good solution into meaningful parts. The recombination operation that is often used is that of averaging two positions or exchanging the coordinates of two potential solutions.

The number of individuals that survive each selection as well as the number of offspring the survivors create are two parameters of the system that must be determined. The other parameters that are somewhat problem-specific are two learning parameters from the mutation operation.

A method of optimisation that is also sometimes included under Evolutionary Computation is Particle Swarm Optimisation (PSO). While most EC techniques are inspired by the evolution of nature in some way, this relatively new technique falls into the category of Swarm Intelligence methods and is based on social interaction in nature. Originally a simulation of the social behaviour of swarming insects similar in some respects to the flocking behaviour described by Reynolds (see section 2.2.2), the swarm's searching was later applied to optimisation [KE95].

This technique, more fully described in the next section, has a number of advantages that make it a good choice for the initial implementation of our Constraint Satisfaction System. A PSO can be easily implemented with few design decisions. The members of the potential solution population are defined as particles in real space and so map directly to the parameter search space of our problem. The social interaction is simpler than that of other EC methods, such as ES, as there is no mutation or recombination, only a limited sharing of knowledge. This element of the PSO is simply defined and there are few parameters to tune in the whole system, most of which have well known default values.

The method that we choose to optimise the objective functions defined by the crowd constraints specified by the user is thus made up of a weighted sum compromise scheme optimised by a PSO. This implementation has the advantage of being simple, able to handle the unknown nature of the objective functions, easily mapped to our problem and parallelisable. The ease of implementation is of importance in our case as this is an attempt to explore the concept of constraint-based behavioural control of crowds. It must be stressed that this is only one attempt to solve this problem. The consideration of alternatives is discussed in section 9.2.

7.4.3 Particle Swarm Optimisation

The Particle Swarm Optimisation (PSO) technique differs from most other evolutionary computation optimisation in that it is inspired by social behaviour in nature as opposed to evolution. Since its introduction, the technique has been much studied and improved. We implement the well tested PSO described by Parsopoulos and Vrahatis [PV02a].

Like other evolutionary computation techniques, the PSO makes use of a set of potential solutions which are updated in cooperation in order to determine the optimal solution. Each member of this population is referred to as a particle in reference to the manner in which it is updated. For a particular scenario in which there are n parameters to be altered, the i^{th} particle's position is given by a vector, $\mathbf{x_i}$ of length n and its velocity by a second vector, $\mathbf{v_i}$, of similar dimension. Each particle also remembers its previous best position, $\mathbf{p_i}$ and b, the index of the particle with the best remembered score. At the q^{th} iteration of the search (denoted by a superscript), the particle is updated as follows:

$$x_i^{q+1} = x_i^q + v_i^q,$$
$$v_i^{q+1} = \chi(w^q v_i^q + c_1 r_1^q (p_i^q - x_i^q) + c_2 r_2^q (p_b^q - x_i^q))$$

where r_1^q and r_2^q are random numbers in the range $[0,1]$ and c_1, c_2 and w_q are constants. The random numbers vary the degree to which the particle is influenced by its own previous best position and the previous best of the whole swarm respectively. The constants control the balance between the global and local search behaviour of the algorithm. χ is a function which limits the velocity to some maximum, v_{max} as follows:

$$\chi(\mathbf{v}) = \begin{cases} \mathbf{v} & \text{if } |\mathbf{v}| \leq v_{max} \\ \frac{v_{max}\mathbf{v}}{|\mathbf{v}|} & \text{otherwise} \end{cases}$$

This system of particles model the social interaction of a swarm of creatures which have memory as well as some form of communication with one another while cooperatively attempting to fulfil some goal. Each particle changes its velocity according to a combination of its own remembered best position and the best position that the group has found. We implement this improved version of the PSO which has been found to be less susceptible to local maxima [KM02]. It is identical except that the global best position is not taken from the whole group, but rather from two random neighbours. Each particle therefore maintains its own best index, b_i, that is chosen to be one of its two neighbours. The velocity update step then becomes

$$v_i^{q+1} = \chi(w^q v_i^q + c_1 r_1^q (p_i^q - x_i^q) + c_2 r_2^q (p_{b_i}^q - x_i^q)).$$

The two constants, c_1 and c_2, directly weight the particle's reliance on its own best and the global best. Later work suggests that these both be set to a value of 0.5 [PV02a] which is the course that we take as these parameters do not seem to be vital to the technique's convergence properties. Of more importance is the choice of w, referred to as the *inertia weight*.

The effect of w is to control the trade-off between the amount of space covered by the search and how carefully the area covered is searched. This is done by changing the portion of a particle's velocity that is carried over from the current iteration to the next. A large value for w will encourage the particles to spread out and explore the space, maintaining a considerable influence from previous velocities. A small value for w will, in contrast, allow the particles to explore the local area, fine-tuning the search of the current area. Based on this behaviour, experiments have shown that the inertia weight should be initially set high and then decreased as the optimisation progresses [PV02a]. Following the values used by Parsopoulos and Vrahatis, we set w to 1.2 initially and decrease it to 0.4 over the time that the PSO executes.

Penalty Function

An important consideration for the application of a PSO to our particular optimisation problem is the constrained nature of the search space. PSOs have been successfully applied to constrained optimisation problems in the past by making use of a penalty function [PV02b]. This function effectively penalises the fitness values of particles that lie outside of the feasible region of the search space.

We implement a simple, constant version of this penalty function by setting the fitness of any particle outside of the feasible region to zero. A particle is outside of the feasible region if it violates any of the parameter constraints imposed by the agent brains and stored in the array of p-constraint objects. This step encourages the particles in the swarm to move back inside the feasible region. Since the objective function does not need to be evaluated for penalised particles, there is some easing of the computational load by checking the p-constraints and only running crowd simulations for particles within the feasible region of the search space.

Initialisation

The manner in which the PSO is initialised is also of importance. The general approach is to randomly initialise the particle swarm across the feasible region of the search space [PV02a]. In our case this is not an entirely trivial procedure due to the shape of the feasible region – defined by the brain's parameter constraints – and the high number of dimensions. As the objective function is expensive we may only make use of a limited number of particles and so it is desirable that these few particles be well distributed over the search space.

Our first, *velocity-only*, approach was to place all of the particles at the initial parameter position with random velocities. As the search space is convex, any point in the space may be reached along a straight line from the initial position and the particles are scattered by their first move. This approach has the desirable property of favouring the initial position as all particles remember it as their initial best position.

Our second means of initialisation, the *random position* approach, attempts to achieve an even distribution of particles over the feasible region. The process, in two dimensions, is shown in figure 7.3. Each particle has its position set by choosing a random vector to move out along from the initial parameter position. The intersection between this vector and the hyperplanes given by each of the constraints is tested in turn and the shortest distance to a constraining plane found. This distance is multiplied by a random value in the range $[0, 1]$ and the particle moved that amount along the random vector. A second random vector is then generated for the particle's velocity. This vector is similarly scaled to have a magnitude of 0.25, one quarter of the length of the normalised domain that a parameter may vary across.

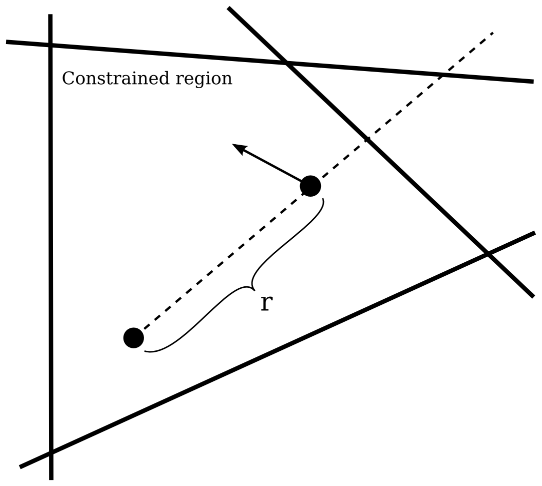


Figure 7.3: *A 2D example of the initialisation of a particle for the PSO. A random vector is chosen to position each particle. The distance from the initial parameters to the border of the feasible space is determined. The particle is placed a random fraction of this distance, r, along the random vector. Once positioned, the particle is given a random velocity scaled by the size of the search space.*

During developmental testing, this random position approach produced better results than velocity-only. One cause of this is likely the random position approach's care in placing the particles such that they do not easily move out of the search space. Using the velocity-only method, a swarm would often leave the search space as the initial energy of the system is all directed outward. Particles outside the search space are influenced only by their memory and shared knowledge and so sometimes skip areas of the search space before re-entering the feasible region as illustrated in figure 7.4. As the number of particles available to a PSO is limited by the expense of function evaluations, the poor use of any particles is highly undesirable. In addition, the random position initialisation removes the radial movement of the velocity-only method and so may better cover the search space.

7.5 The Full Crowd System

The crowd simulation system described in chapter 6 forms the basis for the constraint-based control system that is implemented. This crowd system applies principles from fuzzy control theory in order to produce behaviour that is complex and easily mapped to human thinking and speech. The principles taken from the widely used Massive system are implemented including synthetic hearing

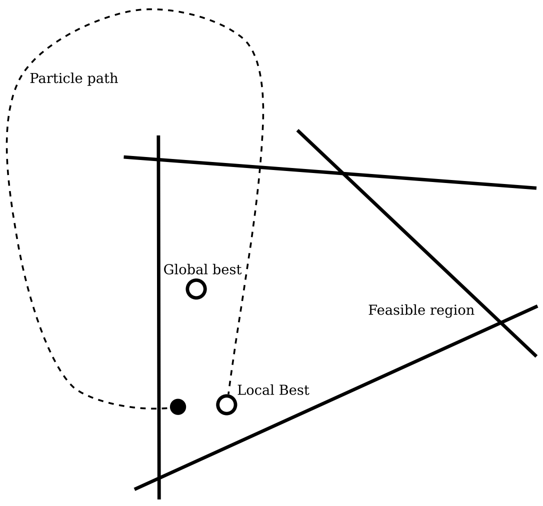


Figure 7.4: *The undesirable path taken by some particles using the velocity-only initialisation approach. The particle is influenced by both its own local best and the global best. The result is that is skips a large part of the search space.*

and vision.

Applied to this crowd simulation system are the constraints discussed in chapter 5. Control is exerted by means of an automatic Constraint Satisfaction System that attempts to meet the user specified constraints by altering the behaviour of the agents in the crowd. This system, shown in figure 7.1, makes use of a Particle Swarm Optimisation (PSO) in order to determine a set of parameters that specify a crowd that meets the particular set of constraints. The PSO makes repeated use of the crowd simulator in order to score different sets of parameters. This score is based on how well the particular crowd meets the constraints. Once a set of parameters is found that meets the constraints specified, the user is presented with a crowd whose behaviour has been altered according to the control applied.

CHAPTER 8

Results and Discussion

The idea of constraint-based control has been presented as well as the details of a "proof of concept" system that represents a possible implementation of such a control mechanism. In this chapter, we describe the testing and performance of our constraint satisfaction system (CSS) according to the design described in section 5.4. We then discuss the testing process and determine the degree to which our proposed method of control succeeds.

8.1 Testing Overview

The aim of the testing strategy is to determine whether the constraint-based control suggested in this dissertation provides additional, useful control of a crowd simulation. In order to determine this, we compare the performance of our system with that of an expert user altering the behaviour of the crowd manually. Both the expert user and the CSS are able change the same brain parameters in order to meet the same set of constraints. The expert user has a complete understanding of the working of the crowd system and, as creator of the original crowd, a good knowledge of the crowd's behaviour. Based on this knowledge we expect the expert user to be able to exert a high level of control over the crowd. By comparing the performance of the expert user to that of the CSS, we are able to judge the quality of the automatic control introduced. The two crowd constraint satisfaction methods, namely expert user and automatic CSS, are compared on the basis of their

- **Ability to meet constraints**
- **Time taken to effect the required change**
- **Degree of perceived change in crowd behaviour**.

These comparisons are made relevant to real-world use by creating eleven different scenarios involving four different agent types. These scenarios are based either on previous use of crowd simulation in visual effects or possible similar uses. Each scenario has a set of constraints that a user might want to apply based on the behaviour of the crowd in the particular scenario.

8.2 Scenarios

The eleven different scenarios used to test the CSS make use of the four agent types shown in figure 8.1. A list of the four agents and a description of their behaviours is provided below:

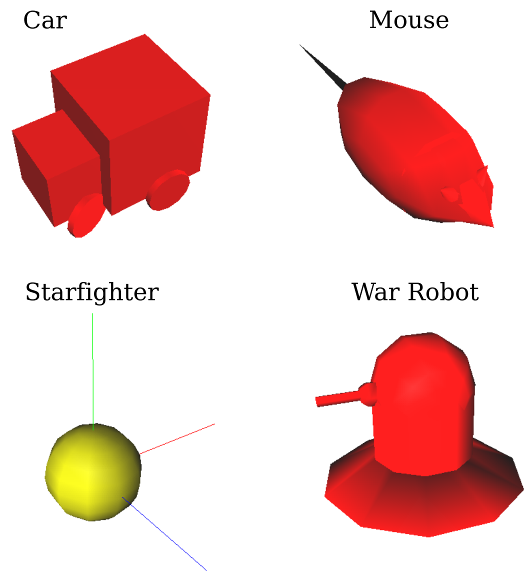


Figure 8.1: *The four different agents used.*

The Car agent is created to race along a track, attempting to keep its speed near a maximum while avoiding other agent and world geometry. The car agent reduces speed as it turns sharply and slows down if a collision is imminent. The brain of the Car agent consists of 40 nodes. Its behaviour is put into effect through two controls: *acceleration*, which may be less than zero in order to model braking, and *turn angle* which may range from $-90°$ to $90°$ and affects the orientation of the agent's *look* vector by rotating it around its *up* vector. This agent's movement is limited to the x-y plane. The agent senses the environment through an *x-geometry-vision* sense and the other agents through a *up-hearing* sense.

The Mouse agent is based on the idea of swarming rodents in movies such as *Ratatouille*. The halting forward motion of a mouse is modelled by the use of the random noise sense to determine whether the mouse should run forward or not. The agent senses the environment by means of the *x-geometry-vision* sense and other mouse agents my means of the *up-hearing* sense. The general behaviour of the mouse agent is similar to that of the Car agent with the agent's movement is limited to the x-y plane. The behaviour is implemented slightly differently with a brain consisting of 30 nodes.

The War Robot agent is designed for use in a battle scenario. This agent has both locomotion and attacking behaviour effected through three control variable, namely, *speed*, *turn angle* and *shoot*. The speed and turn angle affect the agent in a similar manner to the previous two agents. The shoot control models the agent's impulse to shoot and, when set above a particular threshold, the agent will shoot its laser. This shooting is limited by only one laser shot being allowed at a particular time with each laser shot including a cool-down period.

The war robot is slightly more complex than the car or mouse agents with a brain of 42 nodes. This brain makes use of an *up-hearing* sense to sense its allies and, in order to better model vision-based aiming behaviour, an *x-agent-vision* sense to locate its enemies. A control sense allows the agent to take into account the direction that it is turning when approaching a wall or another agent. This agent also moves in only the x-y plane.

The Starfighter agent is able to move in the full 3D world space and has the most complicated brain of the four agents created, consisting of 102 nodes. The agent has a constant speed and is controlled by three orientation altering controls: *rotation-up*, *rotation-right* and *rotation-look*. These three controls describe the degree of rotation attempted around each of the agent's orientation vectors.

The basic behaviour of the agent is to avoid collisions with other starfighters and asteroids while flocking together with the agents in its particular group. These behaviours make use of a *group-heading* sense and *group-centre* sense in order to allow the agent to keep close to its flock-mates as well as two *hearing* senses to avoid its flock-mates and other agents. The agent also has both a *y-vision* and *x-vision* sense that are used to avoid asteroids. Two *target angle* senses are also used in order to allow the agents to flock toward a particular goal.

This agent handles the complexities of free movement in 3D space fairly simply. The agent's brain has rules which work in each plane that an agent may rotate in, attempting to avoid obstacles and head toward the goal. These three behaviours are closely coupled and yet are combined easily by the fuzzy inference and defuzzification process taking place in the brain. The result of these three cooperating behaviours is efficient and plausible steering by the agent with complete freedom in three dimensions.

To Make use of the four agents described above, the following eleven scenarios were constructed. In each case a description of the context for the scenario as well as the constraints applied is included. Screenshots of the initial conditions of each crowd are shown in figure 8.2.

1. **Car Bridge Crossing Scenario.** A group of 10 car agents are initialised in front of a bridge and set to move at maximum velocity. This scenario is inspired by the car chase scenes seen in many action movies. This particular situation proves a challenge to the collision avoidance behaviour of the car agents with a number of collisions occurring in the initial simulation. The basic constraint that a user may want to apply is to prevent collisions between agents or between an agent and world geometry. An additional line crossing constraint was added in order to force the agents across the bridge and so stop the CSS from simply setting all the agents' speeds to zero.
2. **Mouse Bridge Crossing Scenario.** Similar to the Car Bridge scenario described above, this scenario makes use of different agent types. The aim of this scenario is to compare the performance of the CSS in similar situations but with different agent brain structure. In this case 30 mouse agents are required to cross a slightly wider bridge. The constraints applied, as above, are that no agent collides with another agent or world geometry and that all agents cross the bridge.
3. **Car Cornering Scenario.** This scenario was taken from a longer simulation in which four car agents were set to race around a winding track. As the agents approached a particular corner, they were not able to determine the curve of the track and so crashed into each other and the walls. We created this scenario to test the usefulness of the CSS in tweaking a simulation where the expected behaviour fails. The constraints applied to this scenario are to have the agents avoid collisions and to continue around the track in the correct direction.

4. **Car Head-on Crash Scenario.** Two groups of six cars approach each other down a narrow alley. This scenario is inspired by moves such as *The Fast and the Furious*, where groups of cars race at high speed, narrowly avoiding collisions. In the initial set up of this scenario, six out of the twelve cars collided with one another. A typical set of constraints would have these cars avoid all collisions with each other and the alley walls. An additional constraint to force the two different groups out of their respective ends of the alley is added to keep the system from creating cars that make u-turns.

5. **Car Race Scenario.** The constrained flocking work by Anderson *et al.* [AMC03] introduces the idea of controlling an agent so that it comes from the back to win a race. This scenario tests a similar idea. Two groups of cars race along a short course. The first group contains 6 cars while the second group contains only 1 car. The single car of the second group starts behind the first group. In the original run of this scenario, the second group car finishes fifth of seven. The constraint applied is to have the single car of the second group finish ahead of the first group.

6. **Robot War Scenario.** The battle scenes of *Lord of the Rings* changed the way in which crowds are used in visual effects. This scenario is roughly based on the same battle idea. Two groups of war robot agents start with 40 agents on each side. The first group is initialised just behind a "city" that it is defending. The second group of agents starts facing the first, outside of the city limits. The two agents are able to shoot one another and so a battle ensues for the city. This scenario was created to test the use of a hard constraint in a situation where emergence and complex interaction makes direct control difficult. After about 10 seconds of fighting (400 simulation time steps), the number of agents left alive in each group is 14-18. The constraint placed on this simulation is to have between 9 and 11 (inclusive) robots alive in each group after the same time-span.

7. **Escaping Mice Scenario.** This scenario is an attempt to combine different agents in a single scenario. In a scene inspired by war movies, a group of 35 mouse agents are initialised escaping from a city environment. Approaching from outside the city are 5 war robot agents which are able to shoot the mice. The constraint applied to this scenario is similar to that of the Robot War scenario. Initially, in the simulation time interval only one of the mouse agents is left alive while a mouse agent colliding with a war robot agent reduces the robots' numbers to 4. The constraints applied require that 20 mouse agents be left alive and that no agents from either group collide with other agents or the environment. An additional

constraint forcing the mouse agents to escape through the front of the city is applied in order to stop the CSS from allowing the mice to flee in other directions.

8. **Mice Scattering Scenario.** The movie *Vantage Point* saw a crowd of spectators flee from a stadium as the president is assassinated. A number of shots from this scene were created using Massive crowds. The visual effects team that created these shots had considerable difficulty with the closely packed agents colliding with one another and not making it out of the exits[1]. This scenario is loosely based on shots such as this where collisions are difficult to avoid and a large amount of tweaking is required. A group of 100 Mouse agents are initialised in a stadium environment. The agents have an additional brain rule imposed that encourages them to move away from the stage area of the stadium. Initially a large number of collisions occur with 81 agents left alive by the end of the simulation interval (agent state is set to "dead" when a collision occurs). The constraint applied to this scenario prohibits agents colliding with either the environment or each other.

9. **Starfighter Turnback Scenario.** The first of the fully 3D starfighter scenarios is based on science fiction space movies such as *Star Wars*. This scenario has 18 starfighter agents flocking together and with a goal position just behind their initial position. The behaviour of the agents is to turn back towards their goal while attempting to avoid collisions and keep together in a flock. This creates a delicate balance of behaviours that work against one another, resulting in a number of collisions. The constraint applied to this scenario is to have the starfighters avoid collisions with one another. Since an undesirable, but possible, solution is simply to remove the target seeking behaviour, this scenario effectively tests the degree to which the CSS makes undesirable changes to the agents' behaviour.

10. **Starfighter Asteroid-Wall Scenario.** This scenario is reminiscent of science fiction space movies in which space ships fly through asteroid fields. A group of 20 starfighter agents approach an animated asteroid wall through which they must navigate. The initial run of the simulation shows a number of agents colliding with one another or asteroids. The constraint applied is that the agents avoid all collisions while still travelling to the other side of the wall. Since the agents behaviour has no predictive element to it, this task is a difficult one. This scenario tests whether the CSS is able to meet constraints that are somewhat beyond the ability of the agents' behaviour.

[1]http://digitalcontentproducer.com/mil/features/video_step_step_4/

11. **Starfighter Obstacle Scenario.** This last scenario tests the preservation of the flocking behaviour in a group of starfighter agents. The group of 20 agents is faced with an impenetrable wall of asteroids across its path. The flocking behaviour of the group is too strong and so keeps the agents together as they collide with the wall. The constraint applied is that no collisions occur. This scenario tests whether the CSS will maintain the flocking behaviour of the starfighter agents while having them avoid the obstacle.

These scenarios serve as a testbed for our CSS. We believe that this sample, while not exhaustive, is reasonably representative of the space of all possible usage scenarios and so serves to provide insight into the performance of the automatic constraint based control.

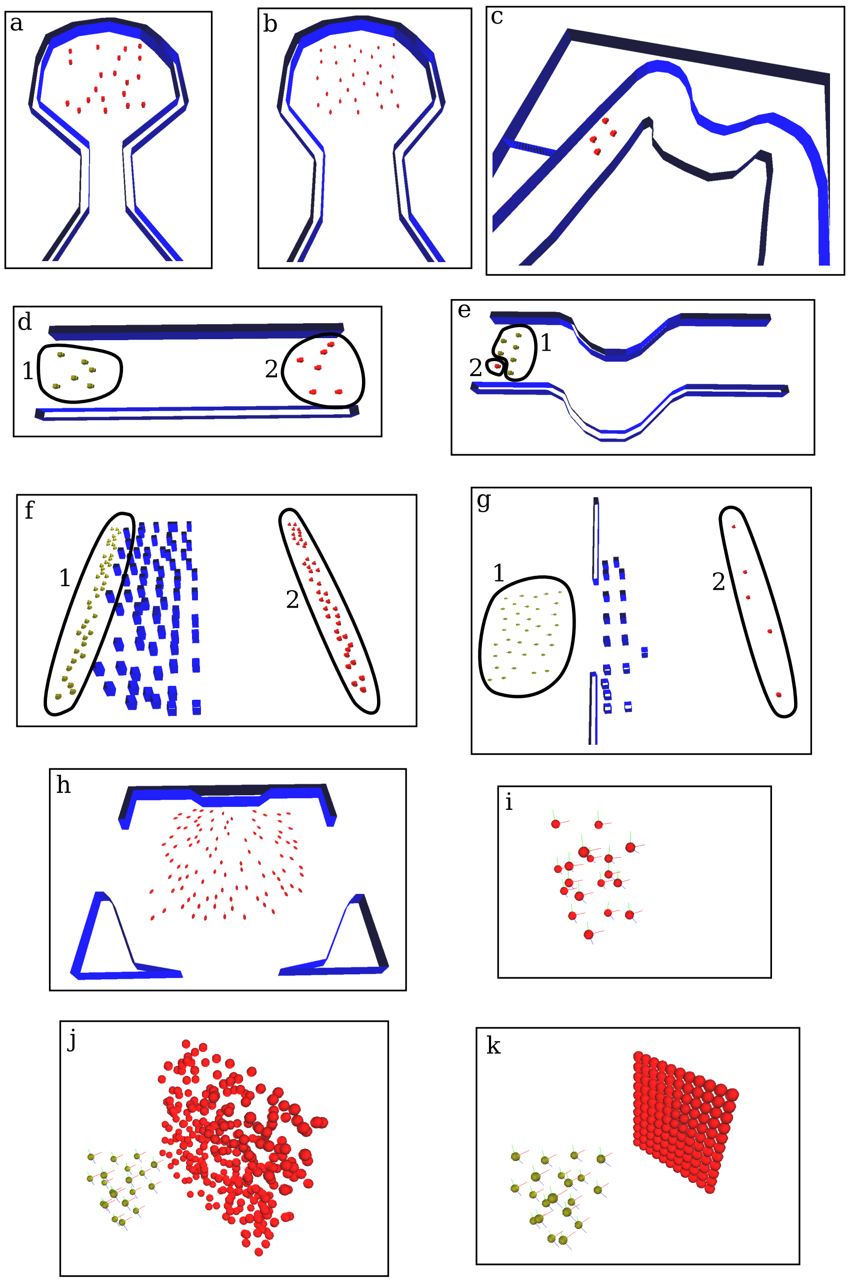


Figure 8.2: *The initialisation of the eleven test scenarios: (a) Car Bridge; (b) Mouse Bridge; (c) Car Cornering; (d) Car Head-on Crash; (e) Car Race; (f) Robot War; (g) Escaping Mice; (h) Mice Scattering; (i) Starfighter Turn Back; (j) Starfighter Asteroid Wall; (k) Starfighter Obstacle. Where more than one group of agents is involved, the different groups are labelled.*

8.3 Constraint Satisfaction Comparison

The first test of the CSS is in its ability to meet arbitrary constraints. As this capability is required in order for the CSS to be useful in controlling crowds, this test is perhaps the most important. We compare the constraint satisfaction abilities of the CSS to that of an expert user who we assume, given enough time, will be able to meet any constraints required. The aim of the CSS is to automatically meet these constraints in less time than the expert user.

For each scenario, the expert user was given the relevant crowd files, a crowd simulation visualisation tool and an agent brain editing tool. The user was then provided with the constraints required and timed as he worked to meet these constraints with no time limit specified. Once the user was confident that the constraints were met, the constraint system was used to measure the actual score produced by the expert user. In determining this final score, the same constraint objects and constraint weights were used as for the subsequent CSS test runs.

As the search process of the CSS is probabilistic, four constraining runs were executed for each scenario. The parameters of the CSS's particle swarm were kept constant throughout the testing process, being set to the defaults described in section 7.4.3. The exception to this was the number of particles which was chosen based on the running time of the simulation as well as the number of parameters. The CSS was allowed to run for a maximum of 2 hours (7200 seconds) on each run with the system continuing to the end of its last improvement iteration when the time limit was reached. If a score of 1 was achieved by any particle, implying all the constraints had been met, the optimisation process was halted early. The CSS was given the same original crowd files as the expert user, as well as a set of constraint objects. This set of constraint objects, with associated weights, was created to match the constraints given to the expert user and used, both to determine the final score of the expert user's attempts, and as input to the CSS. The results of these tests are shown in table 8.1 for comparison.

The scores for the expert user show that, as expected, manual editing of the agents produces a crowd that meets the required constraints in almost all cases. In seven of the eleven scenarios the CSS was also able to completely meet the specified constraints. In the cases where success was not achieved the scores reflect that the constraints were met to a significant extent with the difference usually only a single agent colliding with a wall or failing to move across a line as required.

There are two scenarios for which the score produced by the expert user is sub-optimal. These two exceptions, the Robot War and Starfighter Ast-Wall scenarios, require closer consideration.

Scenario	Expert User Score	CSS Score Average	CSS Score 1	CSS Score 2	CSS Score 3	CSS Score 4
1. Car Bridge Crossing	1	1	1	1	1	1
2. Mouse Bridge Crossing	1	1	1	1	1	1
3. Car Cornering	1	1	1	1	1	1
4. Car Head-on Crash	1	1	1	1	1	1
5. Car Race	1	1	1	1	1	1
6. Robot War	0.9833	0.9896	0.9917	0.9917	0.9833	0.9917
7. Escaping Mice	1	0.9879	0.989	0.9875	0.9875	0.9875
8. Mice Scattering	1	0.9875	0.985	0.99	0.985	0.99
9. Starfighter Turn Back	1	1	1	1	1	1
10. Starfighter Ast-Wall	0.9991	0.9728	0.9728	0.9728	0.9727	0.9728
11. Starfighter Obstacle	1	1	1	1	1	1

Table 8.1: *The constraint satisfaction scores for comparison between the expert user and CSS. The scenarios where the CSS score is lower than the expert user's score are highlighted in grey.*

In the Robot Wars scenario the constraint defined above requires the number of robot agent left alive in each group to be 9, 10 or 11 with no agents colliding with one another or with the environment. The expert user, in altering the crowd, thought that he had met the constraints completely. However, the score reflects two agents, which the expert user did not notice, colliding with the environment. Indeed, it took careful observation and specific highlighting to find these particular agents.

The CSS score for the Robot Wars scenario shows that the automatic system met the constraints to a greater degree than the expert user. The log files of the CSS runs indicate that in all of the four runs the numbers of agents in the final groups were within the tolerance amount. In three of the cases, however, a single agent collided with the environment and in the fourth case a similar collision involving two agents occurred. In this way the CSS was able to meet constraints more exactly than the expert user.

The second scenario in which the expert user does not completely meet the constraints is the Asteroid-Wall scenario. This scenario is extremely challenging in that it involves rapidly moving, densly packed obstacles. Because the agents perform no predictive reasoning, it is difficult for the agents in their current form to avoid such obstacles. The expert user in this case, after attempting the problem for an hour without making progress, decided that the task was impossible. The user's best attempt did not manage to have all the agents reach the far side of the asteroid field with some agents turning and avoiding the asteroids altogether.

The constraint forcing the agents to travel through the asteroid field has a fairly low weighting and so the score achieved by the expert user remains quite high.

The results produced by the CSS show that the system was also not able to meet the constraints specified. When examining the crowd behaviour produced by the automatic system it becomes clear that the CSS did not completely remove the flocking behaviour of the crowd, as was the case with the expert user. With flocking in place, the starfighter agents keep together and some are drawn into the asteroid field. The resulting animations, both of the expert user and the CSS, are also quite different from the original and so we must conclude that the desired result may be attainable only by altering the underlying structure of the agent's brain.

The Escaping Mice scenario is one of the other two scenarios is which the CSS fares worse than the expert user in attempting to meet the specified constraints. When the resulting animations are considered it becomes clear that this scenario is a case where the deeper understanding of the expert user makes a significant difference. The agents produced by the CSS are fairly similar to the original crowd, showing mostly superior collision avoidance. In contrast the expert user meets the constraints largely by reducing the range at which the robot agents first attempt to fire on the mouse agents. This step allows less time for the robots to attack the mice before the end of the simulation time period. The drawback to this approach, shown in the human perception experiment below, is that the crowd behaves distinctly differently. In a separate run in which the CSS was left to search for longer, the automatic system met the constraints completely after 15449 seconds (a little over 4 hours). This crowd appears closer to the original as found by our human test subjects (see section 8.5).

The third scenario in which the CSS is fails to meet the standard of the expert user is the Mice Scattering scenario. The challenge with this scenario is the large number of possible collisions as the mice approach the exits of the stadium. The expert user expressed a large degree of frustration at trying to remove all the collisions as the range the mouse agent's senses is quite short. The method employed by the user was to iteratively find an individual which collided and adjust the brain with only that agent in mind until the particular agent avoided the collision. This process eventually resulted in a collision free simulation. In this case the extra intelligence of the expert user produced better results than the CSS. It should be noted that the numbers of agent collisions in the final simulations produced by the CSS were as few as 2 or 3 out of 100.

In spite of the four cases discussed above, the CSS performed extremely well, altering the crowd such that it completely met the specified constraints in most cases. The comparison with the expert user must take into account the large amount of additional knowledge that is available to the creator of the crowd simulation and the greater understanding of the implication of changes made. Of interest is the consistency of the constraint satisfaction process as there is very little variation from one run the next. In the cases where the system was able to completely satisfy the constraints it did so on every run.

8.4 Timing Comparison

The ability of the CSS to meet constraints placed on crowds is of utmost importance to its usefulness. Given that the system is able to automatically alter crowds in this way, the time frame involved is also of importance. The time taken by the expert user in altering the crowd simulations described in the previous section was recorded and may be compared to the running time of the CSS. The averaged running times are presented in table 8.2. Also included in this table are other relevant details of the test runs.

Scenario	Particles	Approx sim time (sec)	Ave Number of Iterations	Ave Running Time (sec)	Expert User Time Taken(sec)
1. Car Bridge Crossing	300	3.17	27.25	3487.74	2880
2. Mouse Bridge Crossing	300	3.14	8.5	1414.88	1200
3. Car Cornering	700	0.78	1	0	1200
4. Car Head-on Crash	150	1.24	1	0	4020
5. Car Race	300	1.88	70.25	1402.26	1260
6. Robot War	50	27.16	21.5	7889.93	6660
7. Escaping Mice	500	5.43	12.25	7699.41	3420
8. Mice Scattering	200	8.01	66.5	7706.56	1620
9. Starfighter Turn Back	500	0.67	7.75	346.12	840
10. Starfighter Ast-Wall	300	4.99	11	7381.09	3600
11. Starfighter Obstacle	500	3.75	1	0	1020

Table 8.2: *The running times of the CSS compared to the time taken by the expert user. The table gives the number of particles used in the PSO, an approximate running time for the simulation without visualisation, the average number of iterations executed by the PSO in a run, the averaged total running time for a CSS run and the time taken by the expert user in meeting the constraints for each scenario.*

The timing results are mixed, with the CSS outperforming the expert user in four of the test scenarios while recording longer times for scenarios 1, 2 and 5. In scenarios 6, 7, 8 and 10, the CSS did not meet the constraints fully and so ran until the time limit was exceeded. In scenarios 6 and 10, however, the expert user also failed to meet these constraints fully as so perhaps produces an unfair comparison. The three cases in which the average running time of the CSS is recorded as 0 are due to a successful parameter set being found by the initial positioning of the particles in the swarm – timing data was generated over the iterations of the PSO with initialisation not included. An approximation of the time taken for this initialisation can be calculated by multiplying the number of particles by the approximate simulation time. It should also be noted that a crowd simulation is not necessarily run for each particle at each iteration. If a particle is found to be outside of the feasible search space it is given a fitness score of 0 and no crowd simulation is required. This is the reason for the disagreement between the average timing and the multiplication of particle count, simulation time and number of iterations.

A number of factors affect the running time of the CSS. The first is the efficiency of the underlying crowd simulation system. The running times of the simulations shown in table 8.2 are for simulations which do not produce any visualisation. Based on the relatively small numbers of agents involved in these simulations, these timings show that the crowd simulation system itself is rather inefficient – production-level crowd animation software is able to visualise crowds of many thousands in real-time.

Although the crowd simulation implements the successful principles of Massive, it does so in a simple and inefficient manner. In particular, the rendering of an agent's vision and the subsequent fuzzification process, as well as the defuzzification step of each agent's brain, are unoptimised and slow to execute. This is plainly seen in the long running time of the Robot Wars scenario in which each robot agent receives three renderings of the scene at each simulation step.

Another factor which affects the running time of the crowd simulation is the need to constantly update the constraint objects. The implementation of these constraints did not focus on computational efficiency as they are simply example constaints used for testing purposes. Some of the constraints make use of algorithms that are $O(n^2)$ in the number of agents (or worse) and so add considerable computation to each crowd simulation step.

These implementation details should be taken into account when interpreting the timing results of table 8.2. A more full discussion of possible efficiency improvement is included in section 9.2.

8.5 User Evaluation Comparison

Any crowd control system must change the animation of a crowd as required by the controller. An important consideration when applying a means of automatic, high-level control is whether the system changes the crowd's behaviour too much. The believability, and hence usefulness, of a crowd simulation is tied to the behaviour of the individuals. The actions that a crowd member takes must be consistent with his character: an orc must act like an orc and make decisions like an orc. Because of this there is a tension between consistent behaviour and control. The method of control must change the behaviour of the crowd to the extent required but not lose the underlying characteristics of the agent's behaviour.

We address the following question: Does our automatic method of altering agent behaviour in order to meet crowd-level constraints change the underlying behaviour of the crowd to an unacceptable degree?

The current method for altering crowd behaviour is for an expert user to manually change the behaviour of the individual agents in the crowd. Some change in crowd behaviour is necessary in order for the crowd to meet the specified constraints but it is desirable that this be minimal. We assume that an expert user is able to meet the crowd constraints with minimal change to the crowd's behaviour: the user of a crowd system is able to judge the change in the agent's behaviour and may ultimately decide whether the change affected is acceptable.

To produce an acceptable crowd modification, an automatic method of constraint satisfaction should not change the crowd behaviour more than an expert user attempting to meet those same constraints. Since "change in behaviour" is a subjective quality, we conduct an experiment where human subjects supply a *similarity score* when comparing an altered crowd animation to the original.

The comparison is approached on a scenario by scenario basis making use of the same scenarios used for the previous comparisons. For a particular scenario the independent variable is the method of constraint satisfaction. Possible values for this variable are the automatic CSS system proposed in this dissertation and the expert user. The dependent variable is the similarity score in the range $[0, 5]$ with 0 being entirely different and 5 being extremely similar.

In order to obtain an acceptable number of samples, the experiment was run over the internet making use of the social networking site, *facebook*[2]. A "face-

[2]www.facebook.com

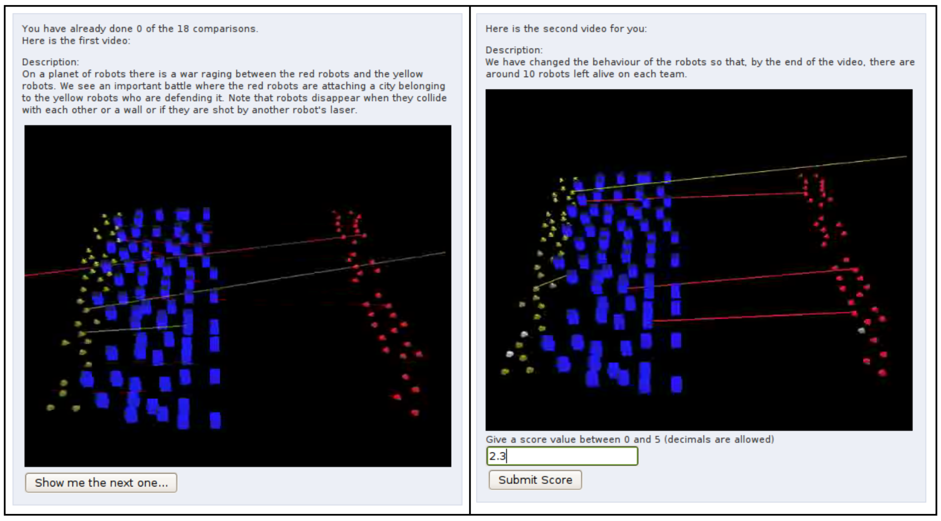
book app" was created that asks the user to perform a number of comparisons between two different crowds shown in short video clips. As the quality being tested is the *manner* in which constraints are met, as opposed to whether they were met or not, only the nine scenarios in which both the CSS and expert user were able to meet the specified constraints equally were used. This was done to avoid biasing the test towards the crowd in which the constraints were met to a lesser degree.

Each comparison consists of a video of the original crowd simulation followed by a simulation of one of the altered crowds. An input field and prompt for a similarity score was given, along with the second video, allowing the test subject to rate how similar the crowds appeared. All the videos for a particular scenario were recorded by capturing the OpenGL frames at the same rate and encoded using identical parameters. The camera position and running times of the simulations were also identical across a particular scenario. The comparisons were given to the test subjects in a random order with no indication of which method was used to alter the crowd simulation in the second video of the comparison. For each comparison, a description of the crowd's context was given with the original video and a description of the constraints applied given with the altered video. Example screen shots are provided in figure 8.3.

This experiment follows a "repeated measures" design with each subject producing a paired sample per scenario. Each pair is made up of the subject's comparison of the original crowd to the human altered crowd and the matching comparison between the original and the CSS altered crowd. The hypothesis that we test is that the similarity score is independent of the method of meeting the crowd constraints. The results of the comparisons are given in table 8.3. These results are somewhat surprising in that they show that, for a number of scenarios, the majority of the subjects tested preferred the crowd generated by the automatic CSS system. In order to judge the significance of these observations, the statistical likelihood of them occurring with our hypothesis being true must be examined.

Each scenario is examined separately. As the samples are paired, statistical analysis is conducted on the difference between the two similarity scores supplied by a particular test subject. The Signed Rank Test [Ric95] is a non-parametric test that may be applied without any assumptions about the underlying distribution of the populations from which the samples are drawn.

If the method of constraint satisfaction makes no difference to the significance score given by a test subject then we would expect the difference in scores to be centred on zero and the differences randomly distributed above and below zero.


You have already done 0 of the 18 comparisons.
Here is the first video:

Description:
On a planet of robots there is a war raging between the red robots and the yellow robots. We see an important battle where the red robots are attaching a city belonging to the yellow robots who are defending it. Note that robots disappear when they collide with each other or a wall or if they are shot by another robot's laser.

Here is the second video for you:

Description:
We have changed the behaviour of the robots so that, by the end of the video, there are around 10 robots left alive on each team.

Give a score value between 0 and 5 (decimals are allowed)

2.3

Figure 8.3: *The comparisons carried out by test subjects in the human study. The image on the left shows the first screen of a user comparison which displays the original crowd. A description of the scenario is included above the video. The image on the right shows the second screen of the comparison where the user is shown the altered crowd with a description of the required change and asked to give a similarity score.*

The first step in applying this test is to discard any pairs in which the scores are equal, resulting in a reduced sample size. Each pair is then assigned a value of either *positive*, for a difference that favours the CSS, or *negative*, for a difference that favours the expert user. If our hypothesis is correct then we would expect the probabilities of a positive and negative result to be equal. Therefore, our null hypothesis is that $p = 0.5$ where p is the probability of a positive result.

As there are only two options for each sample pair: positive or negative, we expect the distribution to follow that of the binomial distribution. We make use of the binomial test which produces the probabilities shown in table 8.4.

There are two ways in which our hypothesis could prove false. Firstly, if the expert user changes the crowd less than the CSS then, in fact, $p < 0.5$. Alternatively, if the CSS changes the crowd behaviour less than the expert user, we have $p > 0.5$. We shall first look at the scenarios in which the scores support $p < 0.5$ and discuss their significance.

For scenarios 2, 3, 4 and 5 we see that a higher number of test subjects preferred the expert user to the CSS. The probabilities of these observations given that the sample is taken from a population with a binomial distribution is still

	Test Subject																									
Scenario	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	1	5	3.5	3	0	2	0.5	3	3	0	1	2	2	4	1.5	4	3	1	2	2	4	1	2	3	3	2
	4	5	3	0	2	3.5	1	5	2	0	1.5	2	2	1	1.5	4	2	4	1	2.5	4	2	2	4	5	2.5
2	4	5	2	4	4	1	1	3	4	0	1	4	2	2	3.8	4	5	2.5	4	3	2	3	3	1	4	4.5
	5	5	1.5	5	3	4.7	2	1	3	0	0.5	1	2	4	1	3	3	2.5	3	3	4	4	1	2	4	4.5
3	5	4	3	4.5	3	4.5	0.5	4.5	5	1	0.5	3	4	5	2.5	3	2	2	4	3	3	4.5	4	3	5	3.5
	3	5	2	3	1	4.9	1	5	4	1	1.5	2	3	2	3.8	4	1	4	4	4	2.5	3	4	4	5	3
4	4	5	2	1	3	4	4	5	5	4	2.5	2	4	3	5	3	2	3	4	4.5	2	3	4	5	5	5
	4	4	1	2	3	4	3	3.5	5	5	2.5	2	4	1	4.5	3	3	2.5	4	5	1.5	3	4	3	5	5
5	0	5	4.75	1	1	0.4	0.5	4	2	0	0.5	0	0	4	0.5	5	0	0	2	1	4	1.5	2	2	4	1.5
	0	5	4	1	1	1	0.5	4.5	1	0	0.2	0	2	4	0	5	0	0	3	1	4.5	0	2	0	4	0
6	0	4	4	1	0	0.4	0.5	3	1	0	0	0	0	5	0.5	5	0	0	1	0	5	0	2	0	4	0
	0	5	4.5	1.5	1	0.2	0.5	4.5	1	0	0.5	0	3	5	0.5	5	0	0	1	3	5	0	1	0	3.5	2
7	0	5	4.75	4	2	1	1	3	1	0	0.5	2	1	3	1.5	5	0	3	0	0	5	3	2	0	4	2.5
	2	5	4	3.5	2	3	1	4.5	2	0	0.5	3	1	4	1	5	0	2.5	1	4	4	3	3	4	3.5	2
8	3	4	2.5	4	2	1.5	1	1	2	1	0.5	2	1	5	0.5	4.5	3	2	3	2	3.5	4.5	1	0	2.7	3
	4	5	2.5	4	3	4.5	1	4.5	2	2	2	2	1	5	0.5	4	3	2	4	1.5	4	4	3	3	3.5	2
9	5	4	2	4	3.5	4.6	4.5	3	4	4	2	4	3	2	4	2	2	1	4	4.5	1.5	4.5	3	4	4.5	3.5
	5	5	3.75	0	3.5	4.9	4	5	3	4	2.5	5	4	2	4	1	4	3	5	5	2.5	4	4	5	4	4

Table 8.3: *The results of the human experiment. The places where the CSS-altered crowd was found more similar to the original, or where the expert user and CSS achieved the same scores, are highlighted in grey.*

fairly high. If we make use of a confidence interval of 95%, none of these four scenarios give us a p-value low enough to reject our null-hypothesis.

For the remaining scenarios: 1, 6, 7, 8 and 9, we observe more test subjects preferring the CSS to the expert user. Using a similar confidence interval of 95% we find that scenarios 1, 6 and 7 do not give us cause to reject our null-hypothesis. However, both the observations for scenario 8 and 9 have a probability of only 0.04 given that $p = 0.5$. We would therefore reject the null-hypothesis in these cases and state rather that $p > 0.5$ which means that people find the CSS to produce less noticeable change than the expert user for these two scenarios. Scenarios 8 and 9 are the Escaping Mice and Robot War scenarios. In both cases the expert user altered the firing range of the War Robot agents in order to meet the required constraints while the CSS did not. This change seems to prove highly noticeable to observers.

Although this surprising result is of interest, it must be taken in the context of all nine tested scenarios. Since the rejection of the null-hypothesis is not widespread, there is not enough evidence to say that the CSS, in meeting specified constraints, changes the crowd's behaviour less than an expert user would. The result does support the use of the CSS, however, as it provides no evidence that the automatic system changes the crowd's behaviour more than necessary.

Scenario	Trials	Successes	Binomial Test
1	18	12	0.12
2	19	8	0.32
3	22	10	0.46
4	13	4	0.13
5	12	5	0.39
6	12	9	0.07
7	17	10	0.31
8	16	12	0.04
9	21	15	0.04

Table 8.4: *The analysis of the human experiment results using the sign test. The Binomial test values give the probability of the observed values given that the method of constraint satisfaction does not affect the similarity score.*

A weakness of the sign test is that it discards the comparisons that result in identical scores. As these observations, in fact, support our null-hypothesis, they are of interest. A statistical test that considers these values is the paired t-test. Our null-hypothesis in this case is that the mean difference between such paired samples is zero. The p-values given by this test are shown in table 8.5. As this test is two-tailed, a confidence interval of 95% does not allow us to reject our null hypothesis for any of the scenarios except the 8^{th}. For this particular scenario, the test scores show that a large number of the test subjects found the crowd generated by the CSS to be more similar to the original. If anything, this result shows that the CSS changes a crowd less than an expert user's manual alteration would.

A note must be made on the limitations of this experiment. An obvious limitation is the finite sample of possible scenarios that we use. This is, unfortunately, unavoidable. Another limitation is due to the crowd simulation system and expert user. To some degree the results of this experiment are dependent on the quality of the brain editing tools and competence of the expert user.

Despite these limitations, the results, taken together, do not give any reason to believe that the CSS changes the behaviour of the crowd *more* than an expert user does. This result quiets our initial concern and supports the use of the automatic crowd control system.

8.6 Discussion

The three comparisons carried out between the CSS and expert user provide a good deal of insight into the possibilities of the proposed constraint-based control. The constraint satisfaction comparison shows that the CSS is able to meet

Scenario	Method	Mean	Variance	Paired t test
1	Expert User	2.25	1.69	0.3
	CSS	2.56	2.17	
2	Expert User	2.95	1.92	0.61
	CSS	2.8	2.27	
3	Expert User	3.35	1.8	0.32
	CSS	3.1	1.78	
4	Expert User	3.62	1.43	0.12
	CSS	3.37	1.51	
5	Expert User	1.79	3.09	0.48
	CSS	1.68	3.49	
6	Expert User	1.4	3.55	0.03
	CSS	1.83	3.77	
7	Expert User	2.09	3.08	0.04
	CSS	2.63	2.27	
8	Expert User	2.32	1.91	0.01
	CSS	2.96	1.66	
9	Expert User	3.39	1.26	0.17
	CSS	3.74	1.67	

Table 8.5: *Statistical analysis of the human experiment using the paired t-test. The p-values in the right-most column give the likelihood of the observed values given that the mean difference between the paired samples is zero.*

the crowd-level constraints in a completely automatic manner with similar effectiveness to the expert user. We discuss the results with particular reference to the design and implementation decision described in chapters 5, 6 and 7.

8.6.1 Constraint-based Control

Our crowd control system makes use of constraints based on their effectiveness in previous work. Although a study of the use of these constraints is not our aim, the results of our system supports the use of constraints as a worthwhile means of applying control. In each scenario that was tested, the required changes were decided before the specification of formal constraints was considered. Despite this, the number of constraints required was not very large, nor were they difficult to determine. In fact, required changes to a crowd simulation are often naturally stated as constraints, even if the specifics are not included.

The CSS makes use of constraint objects that impose few limits on the constraints that they may describe. The constraints used to test the CSS were based entirely on the requirements of the specific scenarios with no consideration for linearity, independence or consistency between constraints. A number of the constraints were, in fact, closely related. For example, the line crossed constraint is dependent to some degree on the agent collision and geometry collision constraints as agent who collide are not able to continue moving across the required line. In the Asteroid-Wall scenario, two conflicting constraints that may not both be satisfied at once were specified: agents avoid collisions and agents move across a line on the far side of the asteroid field. In each case the CSS was able to handle these constraints in a manner similar to the expert user. This flexibility of the constraint objects allows a large degree of freedom to the prospective user.

An aspect of such constraint-based control that must be noted is the need to clearly specify constraints. In the original testing done on the Car Bridge Crossing scenario the only constraint given was for the agents to avoid collisions. The result returned by the CSS on one control run simply reduced the speed of the agents to such an extent that they did not reach the bridge and so did not collide. This particular problem was easily dealt with by the introduction of the line crossing constraint. A more comprehensive usability study may be required in order to determine the usefulness of the constraint control we have produced. Our use of constraints in the control of our test crowds suggests that a relatively small number of standard constraints may be suitable for many different scenarios.

8.6.2 Behavioural Alteration

The control of a crowd simulation by altering the agent's understanding of their behavioural rules has been shown to be effective. In almost all of the test sceshouldnarios both the expert user and the CSS were able to meet the constraints by changing only the membership curves and hedge parameters of the crowd's fuzzy reasoning. The human experiment results show that this method of altering a crowd's behaviour can effectively be carried out automatically without producing undesirable changes in the crowd's behaviour. In addition, this method has the advantage of working well with existing control methods as the changes made to the crowd are similar in nature to those made by an expert user.

8.6.3 Optimisation Method

The use of a PSO with a simple weighting compromise strategy has proved to be an effective choice for optimisation method. Although simple, this combination was able to perform well in our tests, even under difficult circumstances. The search space has a number of properties that make searches difficult and there is no clear method that should be used.

The high dimensionality of the search space and expensive objective function provide a significant challenges to the optimisation technique. Due to the high dimensionality, the particles of the PSO are only sparsely distributed through the space. Added to this is the limited number of iterations that are possible in the CSS's running time. The objective space may be highly discontinuous and there is pressure on the system to make only minimal changes to the parameters – moving as little as possible from the initial position supplied.

Despite all these characteristics of the problem, the CSS implementation was able to meet the required constraints in almost all of the test scenarios. The parameters found proved to introduce no more change in the agent's behaviour than was required by the constraints. As this is a first attempt at implementing such a constraint satisfaction system, there are likely improvements that could be make. This suggests that this form of crowd control is highly effective.

8.6.4 Effectiveness of Implementation

In many of the scenarios, the expert user expressed frustration when attempting to meet the required constraints, since changes to the agent's reasoning affected the crowd in unpredictable ways. The automation provided by the CSS should, therefore, be emphasised. The creation and tweaking of agent brains in the visual effects industry takes a considerable number of man-hours [WWB06]. A system that takes work away from the user by automating the control of a crowd is of value and is precisely the aim of this system, one which is largely met.

It is also highly significant that the CSS produces no more noticable change in crowd behaviour than an expert user meeting a similar set of constraints. The CSS does not have access to the perceptual input available to a human user and so might be expected to change the crowds in undesirable ways. This, however, was not found to be the case. The preservation of the original crowd behaviour may be due to the manner in which the crowd is altered – the membership curves that determine an agent's understanding of its fuzzy rules. Because the underlying stucture of the rules is not changed, the basic behaviour is preserved.

Another possible explanation is inherent in the PSO implementation. Because each particle remembers its best previous position, the inital parameter set is favoured, resulting in the particles moving back toward this starting point. It is also interesting to note that, in most cases, the magnitude of the change in the parameter vector made by the CSS was significantly smaller than that made by the expert user. A human user may tend to make larger changes to parameters in order to produce a visible difference while the random velocities of the PSO sometimes take extremely small steps. This spatial proximity, although not directly linked to perceptual difference, may contribute to the minimal perceptual change that the CSS produces.

A consideration in the usefulness of the provided constraint-based control is the long running times in some of the timing comparisons. These results, however, are produced by an unoptimised system running on a single machine. The pre-visualisation tools provided by Massive allow far larger crowds to be simulated and rendered in real-time and so such a fully-fledged crowd simulation system should allow running times to be considerably reduced. More importantly, the multiple sample nature of the PSO may be easily run in parallel with little communication overhead. This will produce a speed-up that is linear in relation to the number of computing nodes. Our unoptimised system running on even 10 machines would produce running times that are far more useable. As parallel clusters are already common-place in the visual effects industry, this approach could be implemented with little difficulty.

With the running time of the CSS improved by the means described above, the CSS provides an effective mechanism for control of an agent-based crowd. Constraints may be arbitrarily described and the system meets these constraints in a manner which is consistent with the ideas behind their fuzzy logic reasoning. These constraints are met to a degree equivalent to the control applied by an expert user.

CHAPTER 9

Conclusion

In this dissertation we have considered the current state of the art in crowd simulation with particular reference to visual effects. The current systems for creating crowds in film, although impressive, still lack control due to the use of autonomous agents in the simulation of these crowds. We have proposed a constraint-based system which provides a means of high-level control. This system alters the behaviour of the crowd as a whole by changing the reasoning process of the individual agents such that they meet the constraints specified by a user. In this way the overall behaviour and motion of a crowd can be controlled.

The implementation of this constraint-based control method has been built on top of a crowd simulation system that applies the successful principles of the widely used Massive crowd animation package. The Constraint Satisfaction System (CSS), through which the user specified constraints are automatically met, has been compared to the performance of an expert user manually altering the crowd behaviour in a number of scenarios. These tests have shown the CSS to be similar to the expert user in its ability to meet constraints as well as in the degree to which it alters the original crowd's behaviour. The time taken by the CSS to produce these crowds that meet the user constraints is somewhat lengthy but an optimised crowd simulation and parallel search implementation should reduce this time to an acceptable level.

9.1 Application

The application of this additional control mechanism is discussed with reference to its use in visual effects as this is where this research is aimed. The CSS may be used in both the authoring of crowds and the per-shot tweaking of existing crowds to meet specific needs.

The creation of new crowds is a lengthy process in which a significant portion of time is spent defining how the agents in the crowd behave. In the movie *Narnia*, for example, the brains of the Massive agents took nine months to construct. The changes that the CSS makes to the behaviour of agents in a crowd are applied to the "master" agents which are instanced to create large crowds. These changes are made to parameters that might be altered by the crowd animator in creating a crowd. Because of this, the additional automatic control provided by the CSS may be used as part of the authoring process. Crowds that have constraints applied to them may be altered further by hand or other methods.

Due to the complexities involved in creating an agent's brain and motion, crowd packages such as Massive provide ready-made agents that may be used to form crowds "out of the box". This step is effective for creating standard crowds, such as those that fill stadiums or randomly wander in an environment. These packaged crowds, however, are unlikely to behave exactly as required and so some editing of the agent behaviour is required. For a user unfamiliar with the structure and workings of a particular agent's brain, the editing and tweaking of a crowd's behaviour is a challenging task. The control proposed in this dissertation could be applied to such crowds without any knowledge of how their behaviour is actually described or created. In this way, "out of the box" crowds can be easily customised without a lengthy process of first understanding how they work.

9.2 Limitations and Future Work

The manner in which control is applied to the agent crowds is through altering the parameters of their brain nodes. In fuzzy logic terms this is the altering of the membership curves that define the fuzzy sets used in the inference that takes place in an agent's brain. This has the effect of altering an agent's understanding of a particular value used in a fuzzy rule. The CSS is not able to change the structure of the rules, however, which means that there are limitations to the constraints that may be successfully met.

An additional means of meeting particular constraints may be used in cooperation with the approach presented in this dissertation. On such possibility makes use of the random variation often introduced to crowds by a crowd animation system. Each instance of a master agent created in Massive is varied with regard to its possible motion and behaviour. This randomness, which produces more lifelike crowds with realistic variation, was not included in our implemented system. An extension to our system might allow this random element to be used to alter the initial conditions of a simulation such that the required constraints are met. This added control would be useful for per-shot tweaking in which the global changes of our behaviour-altering CSS are not sufficient.

The two main areas in which further research may be conducted are the use of crowd level constraints in control and the optimisation process used by the CSS. The constraints applied by our system are general objects which were created individually as needed. As they stand, these constraint objects require both a knowledge of the crowd simulation system and some programming skill in order to create. This is ideal for use in visual effects as these constraint objects need to be available quickly and used by non-programmers. Research into the usability of these constraints is required in order to determine how this additional control should best be presented to an end-user.

The CSS makes use of a particle swarm optimisation (PSO) with a weighted sum compromise approach in order to optimise the multiple objective functions represented by a user's constraints. There are, however, a number of different options in this area, some of which may produce better results that the system currently implemented. In particular, a multi-objective optimisation method that finds a Pareto-optimal subset of possible compromise solutions may be preferable to a single solution. This would allow a user to decide on the compromise between objectives even during the running of the CSS.

The use of a PSO method in the searching of a particular simulation's parameter space was based on its ease of implementation, potential for parallel computation and ability to be used with black-box functions. There are a number of other possibilities that may be explored in this area, however. An interesting option is the use of some form of meta-modelling technique, discussed in section 7.4, in order to produce a substitute objective function. The parallel implementation of the PSO or some other evolutionary computing method would be of value in order to determine the possible speed-up achieved by such an approach.

One of the main difficulties in searching a simulation's parameter space is the large number of parameters that may be varied. The parameters taken from the brain are done so in a rather naive manner with little consideration for the relationship between them or for those parameters that are unlikely to have any effect on the crowd's meeting the specified constraints. The altering of parameters in a more intelligent manner is, therefore, an area that may produce some fruit if more closely examined.

One of the limitations of our investigation has been the lack of expert users available to evaluate the technique. Valuable insight into the effectiveness and usefulness of this form of proposed control might require the building of a more complete system deployed in industry where animators could use such a system in the animation of fully fledged film crowds.

An additional limitation to our evaluation of the CSS is in the simplicity of the crowd model used. The agents implemented for our crowd simulation system are only able to take a small number of simple actions. The perceived change in the crowd's behaviour could be affected by more complex motion trees and testing on a crowd system with more complex motion is desirable.

Massive agents used in films may have brains containing 7000 or 8000 nodes. In contrast, the agents used for our tests have only around 100 nodes, mostly due to their much simplified motion. A question, therefore, remains as to how well our control system will perform when the complexity of the agent brains is increased. The simple increase in simulation time required by more complex agents can be offset my a more efficient implementation of the crowd simulation. However, an increase in brain nodes will produce a corresponding increase in the number of parameters that may be altered by the CSS.

In order to deal with this, these parameters may be dealt with in a more intelligent may, as described above. It is likely that large sections of the brain can easily be identified as having little to do with particular constraints and so not considered by the optimisation. Most importantly, the parallel nature of the PSO allows additional computational power to be applied in order to improve the performance of the CSS. While an expert user is limited in how fast a crowd may be altered, this automatic system may be made almost arbitrarily fast.

References

[ABL04] Matt Aitken, Greg Butler, Dan Lemmon, Eric Saindon, Dana Peters, and Guy Williams. “The Lord of the Rings: the visual effects that brought middle earth to the screen.” In *SIGGRAPH ’04: ACM SIGGRAPH 2004 Course Notes*, p. 11, New York, NY, USA, 2004. ACM.

[AMC03] Matt Anderson, Eric McDaniel, and Stephen Chenney. “Constrained animation of flocks.” In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 286–297, San Diego, California, 2003. Eurographics Association.

[BG95] Bruce M. Blumberg and Tinsley A. Galyean. “Multi-level direction of autonomous creatures for real-time virtual environments.” In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 47–54. ACM, 1995.

[BH97] David C. Brogan and Jessica K. Hodgins. “Group Behaviors for Systems with Significant Dynamics.” *Autonomous Robots*, **4**(1):137–153, March 1997.

[Blu97] Bruce M. Blumberg. “Go with the flow: synthetic vision for autonomous animated creatures.” In *Proceedings of the first international conference on Autonomous agents*, pp. 538–539, Marina del Rey, California, United States, 1997. ACM.

[Che04] Stephen Chenney. “Flow tiles.” In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 233–242, Grenoble, France, 2004. Eurographics Association.

[CL08] J.Y. Chang and T.Y. Li. “Simulating Virtual Crowd with Fuzzy Logics and Motion Planning for Shape Template.” In *Proceedings of the IEEE International Conference on Cybernetics and Intelligent Systems*, 2008.

[DHR93] Dimiter Driankov, Hans Hellendoorn, and Michael Reinfrank. *An introduction to fuzzy control.* New York Springer-Verlag, 1993.

[Dun02] Jody Duncan. “Ring Masters.” *Cinefex*, (89):64–131, April 2002.

[FTT99] John Funge, Xiaoyuan Tu, and Demetri Terzopoulos. “Cognitive modeling: knowledge, reasoning and planning for intelligent characters.” In *Proceedings of the 26th annual conference on Computer graphics* *and interactive techniques*, pp. 29–38. ACM Press/Addison-Wesley Publishing Co., 1999.

[GKM01] Siome Goldenstein, Menelaos Karavelas, Dimitris Metaxas, Leonidas Guibas, Eric Aaron, and Ambarish Goswami. "Scalable nonlinear dynamical systems for agent steering and crowd simulation." *Computers & Graphics*, **25**(6):983–998, December 2001.

[Gri03] Kim Griggs. "Assault on the Senses." *IEE Review*, **49**(3):24, March 2003.

[HBJ05] Dirk Helbing, Lubos Buzna, Anders Johansson, and Torsten Werner. "Self-Organized Pedestrian Crowd Dynamics: Experiments, Simulations, and Design Solutions." *TRANSPORTATION SCIENCE*, **39**(1):1–24, February 2005.

[HM95] Dirk Helbing and Péter Molnár. "Social force model for pedestrian dynamics." *Physical Review E*, **51**(5):4282, May 1995.

[Hug03] Roger L. Hughes. "The Flow of Human Crowds." *Annual Review of Fluid Mechanics*, **35**:169–182, 2003.

[KE95] J. Kennedy and R. Eberhart. "Particle swarm optimization." In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pp. 1942–1948 vol.4, 1995.

[Ker05] Laurent Kermel. "Crowds in Madagascar." In *ACM SIGGRAPH 2005 Courses*, p. 6, Los Angeles, California, 2005. ACM.

[KGP02] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. "Motion graphs." *ACM Trans. Graph.*, **21**(3):473–482, 2002.

[KLL08] Taesoo Kwon, Kang Hoon Lee, Jehee Lee, and Shigeo Takahashi. "Group motion editing." *ACM Trans. Graph.*, **27**(3):1–8, 2008.

[KM02] J. Kennedy and R. Mendes. "Topological Structure and Particle Swarm Performance." In *Proceedings of the Congress on Evolutionary Computation*, May 2002.

[KS02] Ansgar Kirchner and Andreas Schadschneider. "Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics." *Physica A: Statistical Mechanics and its Applications*, **312**(1-2):260–276, September 2002.

[LCF05] Yu-Chi Lai, Stephen Chenney, and ShaoHua Fan. "Group motion graphs." In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 281–290, Los Angeles, California, 2005. ACM.

[LCH07] Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. "Group behavior from video: a data-driven approach to crowd simulation." In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 109–118, San Diego, California, 2007. Eurographics Association.

[LD04] Fabrice Lamarche and Stphane Donikian. "Crowd of Virtual Humans: a New Approach for Real Time Navigation in Complex and Structured Environments." *Computer Graphics Forum*, **23**(3):509–518, 2004.

[LK06] Manfred Lau and James J. Kuffner. "Precomputed search trees: planning for interactive goal-driven animation." In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 299–308, Vienna, Austria, 2006. Eurographics Association.

[LMM03] C. Loscos, D. Marchal, and A. Meyer. "Intuitive crowd behavior in dense urban environments using local laws." In *Theory and Practice of Computer Graphics, 2003. Proceedings*, pp. 122–129, 2003.

[LYW03] G.P. Liu, J.B. Yang, and J.F. Whidborne. *Multiobjective Optimisation and Control*. Engineering Systems Modelling and Control. Research Studies Press Ltd., 2003.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. "Particle-based fluid simulation for interactive applications." In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[MH04] Ronald A. Metoyer and Jessica K. Hodgins. "Reactive Pedestrian Path Following from Examples." *The Visual Computer*, **20**(10):635–649, December 2004.

[NG03] C. Niederberger and M. Gross. "Hierarchical and Heterogenous Reactive Agents for Real-Time Applications." *Computer Graphics Forum*, **22**(3):323–331, 2003.

[NRT95] Hansrudi Noser, Olivier Renault, Daniel Thalmann, and Nadia Magnenat Thalmann. "Navigation for digital actors based on synthetic vision, memory, and learning." *Computers & Graphics*, **19**(1):7–19, 1995.

[PAB07] N. Pelechano, J. M. Allbeck, and N. I. Badler. "Controlling individual agents in high-density crowd simulation." In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 99–108, San Diego, California, 2007. Eurographics Association.

[Pat06] John Andre Patterson. *Implementing autonomous crowds in a computer generated feature film.* PhD thesis, Texas A&M University, April 2006.

[PG96] Ken Perlin and Athomas Goldberg. "Improv: a system for scripting interactive actors in virtual worlds." In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 205–216. ACM, 1996.

[PHM06] Julien Pettr, Pablo de Heras Ciechomski, Jonathan Mam, Barbara Yersin, Jean-Paul Laumond, and Daniel Thalmann. "Real-time navigating crowds: scalable simulation and rendering." *Computer Animation and Virtual Worlds*, **17**(3-4):445–455, 2006.

[PJM08] Simon Perkins, David Jacka, Patrick Marais, and James Gain. "A Spatial Awareness Framework for Enhancing Game Agent Behaviour." In *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games*, pp. 15–22, 2008.

[PKL08] Julien Pettré, Marcelo Kallmann, and Ming C. Lin. "Motion planning and autonomy for virtual humans." In *ACM SIGGRAPH 2008 classes*, pp. 1–31, Los Angeles, California, 2008. ACM.

[PPD07] Sbastien Paris, Julien Pettr, and Stphane Donikian. "Pedestrian Reactive Navigation for Crowd Simulation: a Predictive Approach." *Computer Graphics Forum*, **26**(3):665–674, 2007.

[PV02a] K.E. Parsopoulos and M.N. Vrahatis. "Recent approaches to global optimization problems through Particle Swarm Optimization." *Natural Computing*, **1**:235–306, 2002.

[PV02b] Konstantinos E. Parsopoulos and Michael N. Vrahatis. "Particle swarm optimization method for constrained optimization problems." In *In Proceedings of the Euro-International Symposium on Computational Intelligence 2002*, pp. 214–220. IOS Press, 2002.

[RBP98] Enrique H. Ruspini, Piero P. Bonissone, and Witold Pedrycz, editors. *Handbook of Fuzzy Computation.* Institute of Physics Publishing, Bristol and Philadelphia, 1998.

[Ree83] W. T. Reeves. “Particle Systems—a Technique for Modeling a Class of Fuzzy Objects.” *ACM Trans. Graph.*, **2**(2):91–108, 1983.

[Rey87] Craig W. Reynolds. “Flocks, herds and schools: A distributed behavioral model.” In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 25–34. ACM, 1987.

[Ric95] John A. Rice. *Mathematical Statistics and Data Analysis.* Duxbury Press, Wadsworth Publishing Company, Belmont, California, 2nd edition, 1995.

[Rob08] Barbara Robertson. “Mummy’s the Word.” *Computer Graphics World*, **31**(8):14–20, August 2008.

[RSC99] Edmund M. A. Ronald, Moshe Sipper, and Mathieu S. Capcarrère. “Design, Observation, Surprise! A Test of Emergence.” *Artificial Life*, **5**(3):225–239, Summer 1999.

[SGC04] Mankyu Sung, Michael Gleicher, and Stephen Chenney. “Scalable behaviors for crowd simulation.” *Computer Graphics Forum*, **23**(3):519–528, 2004.

[SKG05] Mankyu Sung, Lucas Kovar, and Michael Gleicher. “Fast and accurate goal-directed motion synthesis for crowds.” In *SCA ’05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 291–300, New York, NY, USA, 2005. ACM.

[ST05] Wei Shao and Demetri Terzopoulos. “Autonomous pedestrians.” In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 19–28, Los Angeles, California, 2005. ACM.

[TCP06] Adrien Treuille, Seth Cooper, and Zoran Popović. “Continuum crowds.” In *ACM SIGGRAPH 2006 Papers*, pp. 1160–1168, Boston, Massachusetts, 2006. ACM.

[TdB02] Edward Tunstel, Marco A. A. de Oliveira, and Sigal Berman. “Fuzzy behavior hierarchies for multi-robot control.” *International Journal of Intelligent Systems*, **17**(5):449–470, 2002.

[THL04] Daniel Thalmann, Christophe Hery, Seth Lippman, Hiromi Ono, Stephen Regelous, and Douglas Sutton. "Crowd and group animation." In *ACM SIGGRAPH 2004 Course Notes*, p. 34, Los Angeles, CA, 2004. ACM.

[TSK07] W Tantisiriwat, A Sumleeon, and P Kanongchaiyos. "A Crowd Simulation Using Individual-Knowledge-Merge based Path Construction and Smoothed Particle Hydrodynamics." In *Proceedings of 15-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2007*, University of West Bohemia, Czech Republic, February 2007.

[TT94a] Xiaoyuan Tu and Demetri Terzopoulos. "Artificial fishes: physics, locomotion, perception, behavior." In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 43–50. ACM, 1994.

[TT94b] Xiaoyuan Tu and Demetri Terzopoulos. "Perceptual modeling for the behavioral animation of fishes." In *Proceedings of the second Pacific conference on Fundamentals of computer graphics*, pp. 185–200, Beijing, China, 1994. World Scientific Publishing Co., Inc.

[UHT04] Branislav Ulicny, Pablo de Heras Ciechomski, and Daniel Thalmann. "Crowdbrush: interactive authoring of real-time crowd scenes." In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 243–252, Grenoble, France, 2004. Eurographics Association.

[WSW04] L. Wang, S. Shan, and G.G. Wang. "Mode-pursuing sampling method for global optimization on expensive black-box functions." *Engineering Optimization*, **36**(4):419–438, August 2004.

[WWB06] Dean Wright, Bill Westenhofer, Jim Berney, and Scott Farrar. "The visual effects of The Chronicles of Narnia: the Lion, the Witch and the Wardrobe." *Comput. Entertain.*, **4**(2):4, 2006.

[Zad65a] L. A. Zadeh. "Fuzzy sets." *Inf. & Contr.*, **8**:338–353, 1965.

[Zad65b] L. A. Zadeh. "Fuzzy sets and systems." In *Proc. Symp. Syst. Theory*, pp. 29–37, 1965.