Exploring the real world applications of cellular automata and its application to the simulation of flocking behaviour

by

James Shannon, BSc

A dissertation presented to the University of Dublin,

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

(Interactive Entertainment Technology)

University of Dublin, Trinity College

August 2013

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work, and has not been submitted as an exercise for a degree at this or any other university.

James Shannon

August 29th, 2013

Acknowledgements

I would like to thank my supervisor Dr. Mícheál Mac an Airchinnigh and the course director Dr. John Dingliana for their advice and helpful feedback throughout the academic year. I would also like to thank my family for their support throughout the year.

James Shannon

University of Dublin, Trinity College,

August 2013

Exploring the Real World Applications of Cellular Automata and its application to the Simulation of Flocking Behaviour

James Shannon

University of Dublin, Trinity College, 2013

Supervisor: Dr. Mícheál Mac an Airchinnigh

Flocking behaviour is seen all around us in nature. Birds flock in the sky, fish flock in the oceans and insects flock and swarm underground. Travelling in flocks provides many advantages for those who partake, these include improved chances of finding food while foraging, easier time finding a mate and most importantly, increased protection from predators.

The first algorithm used in the computer aided simulation of flocking behaviour was the Boids flocking algorithm developed by Craig Reynolds in 1987 [1]. This algorithm has allowed computers to simulate how flocks of autonomous agents react to their surroundings and collectively exhibit behaviour which is much more complex and interesting than the individual agents are capable of. While flocking has been simulated for almost three decades it remains a computationally expensive problem which also scales badly as the number of agents in a given simulation increases.

In this dissertation a new approach has been considered in order to tackle the inefficiency problem which prevents flocking behaviour from being fully utilized in real time applications such as video games. This new approach will take advantage of two-dimensional cellular automata which due to its inherent nature of producing complex behaviour from simple, discrete rule sets is a good candidate on which to develop a new flocking technique.
Keeping in mind the increased demand for more complex physical simulations and special effects in real time applications it is the main aim of this dissertation to explore the application of cellular automata to the simulation of flocking behaviour. It is hoped that the new approach will be able to approximate flocking in a similar manner to the Boids algorithm while being efficient enough to use in real time applications. The Boids algorithm will also be implemented on order to provide a benchmark against which to test the new approach.

Contents

3 List of Tables

4 List of Figures

5 Introduction

5.1 Motivation

Flocking is a complex problem, a process which allows autonomous agents to interact with their immediate surroundings and neighbouring agents to produce complex, emergent behaviour can be very computationally expensive depending on the size of the flock. As a result of this expense, flocking is rarely used in real time applications such as video games, instead we see it used as a pre-rendered visual effect in large budget movies such as The Lion King [17] and Batman Returns [18]. With the demand for increasingly complex effects and features in video games it is important that the inefficiency of flocking be addressed and a new approach be considered so that flocking may be fully utilized in real time applications with minimal expense.

5.2 Goals

The goal of this paper is to explore the uses of 2 dimensional cellular automata as a means of lowering the costs of flocking behaviour for use in real time applications. The main goals of the dissertation are as follows:

- Identify key characteristics in flocking behaviour and break them down into their simplest form.
- Convert these basic characteristics from a continuous to a discrete approach to flocking.
- Investigate the application of the new rule set to 2 dimensional cellular automata discrete space.

- Develop a new cellular automata that produces an acceptable approximation of flocking behaviour in discrete form similar to an existing continuous approach, in this case the Boids algorithm.
- The new approach must be efficient enough for use in real time applications.
- The approach must also scale well under dynamic scenes of increasing complexity.

5.3 Dissertation Outline

The state of the art chapter will describe in detail the concept of flocking behaviour in nature and how we go about defining it. The Boids flocking algorithm which was the first flocking algorithm developed for computing purposes and has been extremely important for the field of computer simulated flocking behaviour is also described in detail.

The leading optimization for the Boids algorithm, Bin-Lattice Spatial Subdivision is also discussed in some detail and highlights performance improvements it introduces over the basic un-optimized version of Boids. It is important to remember that the implementation of Boids we use as a benchmark for the purposes of this paper is un-optimized in order to keep performance testing fair.

Following the description of the Boids algorithm is a detailed description of both elementary and two dimensional cellular automata the understanding of which is paramount for this new approach to flocking. Similarities between how we define flocking behaviour and how cellular automata produces complex behaviour are also discussed and the reasons cellular automata can be considered a viable option upon which to base a new flocking technique are explained.

A well-known two dimensional cellular automata known as Conway’s Game of Life is also discussed in order to show how two dimensional cellular automata can exhibit complex, emergent behaviour from a simple set of rules applied to on a large scale.

The research approach chapter explains the initial approach taken to address the problem at hand, in order to break down the problem for application to cellular automata a list of relevant research papers dealing with practical applications of cellular automata was compiled for research into the processes they used to approach their respective problems..

The implementation chapter will explain the in detail new algorithm which has been developed, it will be explained step by step with the aid of several diagrams. It will also show how the algorithm approximates flocking with respect to the three basic principles that the Boids algorithm follows and how it does so in discrete space as opposed to the continuous space which Boids works in.

The testing and evaluation chapter will discuss the conditions under which the Boids and cellular automata approaches were tested against each other. The results of these performance tests have been compiled into a set of graphs which show the performance of both approaches running inside a dynamic scene with varying levels of complexity. The section will explain in detail the main experiments which were run on both approaches and discuss the results of said experiments. There is also an evaluation of the final algorithm which is a review of what has been accomplished with the development of this new approach.

Finally the conclusion chapter will discuss how each of the stated goals for the project were met while the chapter on future work will explain several future applications that the new algorithm may be used in.

6 State of the Art

6.1 Overview of Flocking Behaviour

The concept of flocking as one of the most complex and interesting aspects of behaviour in nature and has long been contemplated and studied. The mechanics by which discrete agents such as birds, insects and fish interact with each other to create fluid, unified motion are still somewhat of a mystery to the scientific community although they have been approximated and modelled by numerous algorithms. One of the most intriguing aspects of flocking behaviour is the illusion that there exists a guiding hand or 'centralised controller' in a given flock when in actuality it is nothing more than the simple behaviours of many individual agents combining to form more complex overall behaviour. In order to develop a new technique for simulating this behaviour we must first think about how we define what flocking behaviour is. We can do this by describing the characteristics of a flock and the individual agents that make it what it is, for any given flock the following traits can be observed,

- It is made up of self-organising autonomous agents
- It displays emergent behaviour
- Flock leaders can emerge and have more influence over the immediate formation of the flock, these may tire and move towards the back of the flock allowing new leaders to come forward.
- It reacts to both inner and external forces in order to maintain structure and reach its destination.

The simulation of flocking behaviour in real time computer applications has been done for decades, algorithms have been developed which attempt to mimic the relationships between autonomous agents in flocks and swarms in order to produce large complex formations such as those seen in nature. The first and most well-known of these algorithms is the Boids flocking algorithm developed by Craig Reynolds which is discussed in detail in the next section. The algorithm was originally used to simulate flocks of birds, but the model has been applied to other problems as well, such as simulating a swarm of bats in the movie Batman Returns (1992) and the iconic wildebeest stampede scene in the movie Lion King (1994).

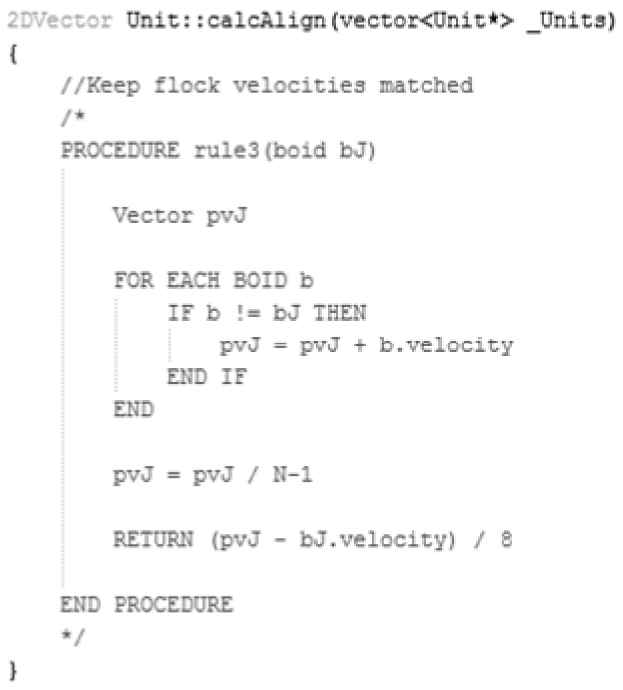
6.2 Boids Flocking Algorithm

The Boids flocking algorithm was developed as a means for describing the relationship between autonomous agents such as fish and birds which when working in close proximity with one another form large complex structures known as flocks. The basic Boids flocking model consists of three simple steering principles [11] which describe how individual boids move based on the positions and velocities of the neighbours which intersect with their cone of sight i.e. the immediate area around them in which they can sense other entities. In a seminal paper on the subject; Reynolds describes this emergent behaviour as “an independent actor that navigates according to its local perception of the dynamic environment”. The 3 principles used in order to accurately describe the behaviour are known as separation, cohesion and alignment. Separation is the principle which allows boids to steer away from their nearest neighbours (within a predefined threshold) in order to avoid overcrowding and collisions in the flock. Alignment is used to allow individual agents to normalise their headings based on the average heading of the agents in its neighbourhood, this is done by each agent matching the average velocity of the other agents that it can interact with. The third principle is cohesion which is used to keep the flocks structure intact by allowing agents to steer towards the average position of surrounding agents there by keeping them together. This cohesion can be overridden by larger forces intervening such as a static obstacle forcing the flock to break up in order to avoid it.

The basic, un-optimized version of the algorithm works as follows,

- Each boid is given access to the entire scene or application space.
- In order for the algorithm to work each boid must define its own local neighbourhood or cone of vision and act only on the behaviour of the boids within that area. These neighbourhoods have a maximum radius which is defined as a distance value from the centre of the boid and an angle (usually based on its flight direction).
- All boids outside the current boids cone of vision are completely ignored by the boid, this applies for all boids when it is their turn to update their states.
- Using the 3 principles described above each boid will interact with the boids in its neighbourhood, the result of this interaction is the formation of a flock of boids or several smaller flocks depending on the forces being applied. These flocks can move towards destinations such as food sources or roam freely while internal forces keep their formations intact.

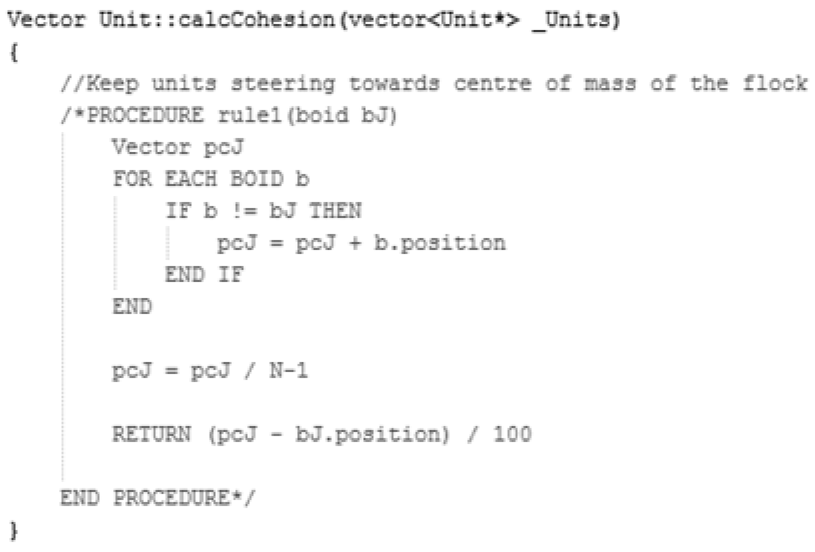
Examples of basic implementations of the 3 principles can be seen in pseudo code below,



```
2DVector Unit::calcAlign(vector<Unit*> _Units)
{
    //Keep flock velocities matched
    /*
    PROCEDURE rule3(boid bJ)

        Vector pvJ

        FOR EACH BOID b
            IF b != bJ THEN
                pvJ = pvJ + b.velocity
            END IF
        END

        pvJ = pvJ / N-1

        RETURN (pvJ - bJ.velocity) / 8

    END PROCEDURE
    */
}
```

Figure 2.1

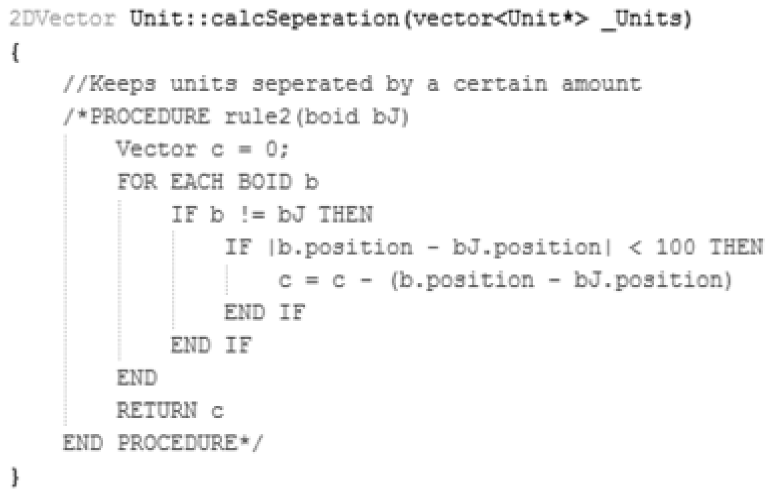
Alignment: This rule is used to make sure the flock travels at a uniform velocity, to achieve this each boid will look at its neighbours and average their velocities. Once this has been done we then divide the velocity by a damping weight, in the above case 8, which can be tuned to provide the best possible results and modify the intensity of this particular force to exert more or less influence over internal flock structure as we see fit.



```
Vector Unit::calcCohesion(vector<Unit*> _Units)
{
    //Keep units steering towards centre of mass of the flock
    /*PROCEDURE rule1(boid bJ)
        Vector pcJ
        FOR EACH BOID b
            IF b != bJ THEN
                pcJ = pcJ + b.position
            END IF
        END

        pcJ = pcJ / N-1

        RETURN (pcJ - bJ.position) / 100

    END PROCEDURE*/
}
```

Figure 2.2

Cohesion: Each boid will look at all boids in its neighbourhood and average all of their position vectors. Again we can divide the result by a damping weight, in this case 100, to scale the forces' influence. This principle will allow the flock to maintain its overall shape and appear more realistic as the boids will attempt to fly inward thus forcing the flock to stay together in formation.



```
2DVector Unit::calcSeperation(vector<Unit*> _Units)
{
    //Keeps units seperated by a certain amount
    /*PROCEDURE rule2(boid bJ)
        Vector c = 0;
        FOR EACH BOID b
            IF b != bJ THEN
                IF |b.position - bJ.position| < 100 THEN
                    c = c - (b.position - bJ.position)
                END IF
            END IF
        END
        RETURN c
    END PROCEDURE*/
}
```

Figure 2.3

Separation: For this principle each boid will look at its neighbours and using a simple Euclidean distance check combined with a threshold we can make the boid steer away from any neighbours that are too close. The correct direction to move is calculated by subtracting the neighbour's position from the current boids position and again if we wish we can apply damping. Damping separation forces is rarely done however as it increases the chance that a mid-air collision will occur which would be catastrophic to the flock.

An interesting feature to note about this algorithm is that while motion in small time steps is predictable and easy to extrapolate, due to its somewhat chaotic nature it is impossible to figure out which direction a flock will be moving over a larger time step.
While the algorithm does produce fairly high quality results there are several performance issues with it, these are as follows:

- It can be quite inefficient as depending on the complexity of the scene as every agent needs to search all other agents in order to know which ones fall into their respective cones of vision so they can take them into account in their calculations.
- The amount of calculations needed grows rapidly as the number of in the simulation scales up, this happens at a rate of O (n_2) in terms of algorithmic complexity meaning that as more boids enter the scene the number of searches each boids needs to perform increases exponentially.

6.3 Optimizing Boids

While the Boids algorithm does provide impressive results it can be somewhat inefficient depending on its implementation, this is mainly due to the O (n_2) complexity of the traversal algorithm each agent in crowd uses to decide for itself which neighbours fall into its environment or 'neighbourhood'. In order to optimize the algorithm a technique known as Bin-Lattice Spatial Subdivision has been utilised. This technique allows much more complex flocks made up of hundreds or even thousands of agents to be simulated in complex environments which contain static and dynamic obstacles to interact with.
This is accomplished by breaking the application space down into easier to handle, discrete sections which vastly reduces the search space of each agent in the flock and thus saving a

massive amount of processing power. Overwhelming agent position querying costs are overcome by pre-sorting the agents in a spatial data-structure [5] or ‘spatial database’ based on their locations in the application space. Research into spatial subdivision has also yielded further improvements for the algorithms optimization through the use of simple heuristics [2].

A second possible optimisation technique is the probabilistic sphere event algorithm (PSEA) developed by Zoran Popovic in his 1994 paper “The Rapid Simulation of Proximal-Interaction Particle Systems” [6]. This is an algorithm based on probabilistic sphere events for simulation of particle systems with particles correlated at small distances. In his conclusion Popovic states that “This algorithm can be applied to many molecular simulations, particle collision detection, and other applications in physically-based modelling, and scientific simulations.” This algorithm has been proven both theoretically and practically to be an improvement over spatial subdivision techniques which were shown to represent the same result as a worst case PSEA algorithm result. The PSEA algorithm also demonstrates an explicit trade-off between speed and accuracy allowing it to be fine-tuned for specific needs on a problem to problem basis.

6.4 Overview of Cellular Automata

The concept of cellular automata was originally developed in the 1940s by Stanislaw Ulam and John Von Neumann while they were colleagues at the Los Almos National Laboratory. While cellular automata was studied to some degree during the 1950s and 1960s, it was not until the development of Conway’s Game of Life, a two-dimensional cellular automaton developed during the 1970’s came along that interest in the subject increased beyond the academic world. The field of cellular automata was brought into the spotlight when Stephen Wolframs seminal book “A New Kind of Science” [7] which covers in incredible detail the field of cellular automata and its applications was published in 2002. This book made controversial claims that cellular automata could have practical applications in area such as processing, modelling of lattice gases and even cryptography.

Any given Cellular Automata can be defined as a discrete model, each consists of a grid of cells each with a finite number of states, usually defined as on or off. The grid of cells may also be called a Moore Neighbourhood or alternatively as a von Neumann neighbourhood, they can be expressed in any number of finite dimensions and can be finite or infinite in size depending on their implementation. Cellular automata are studied in a wide variety of fields including but not limited to computability theory, mathematics and theoretical biology. Cellular automata are considered to be Turing complete when given the right rule sets, meaning that theoretically, they can be used to perform any computations that any single taped Turing machine is able to do given enough time.

The most basic form of cellular automata are one dimensional, where each cell has two possible states, on and off, in this form a cell's neighbourhood is defined as the cell to the left and to the right of it , in terms of a one-dimensional array it would look like this (i-1, i+1) with “i” denoting the position in the array of the cell having its state updated., together with the current cell they make up a neighbourhood of 3 cells and a possible 2^3 (8) states. There are 2^9 (512) rules which can be applied to two dimensional simulations which make use of Moore neighbourhoods, since there are 8 possible configurations for a cell and its two neighbours we can calculate 256 as the maximum possible number of elementary cellular automata, rules are explained in greater detail in the next section.

Stephen Wolfram in his book A New Kind of Science defined four classes into which one can specify cellular automata, they range from type 1 which are the most rudimentary form to type 4 which are much more complex and it has been theorised that some if not all of them are all capable of universal computation.

The class criteria are as follows,

Class 1: Nearly all patterns evolve quickly into stable, homogenous patterns. Any of the initial randomness exhibited within these pattern disappears.

Class 2: Nearly all patterns evolve into either stable or oscillating structures, some of the patterns initial randomness may dissipate but a small part of it will usually remain, also any local changes to the initial pattern which occur will remain local.

Class 3: Nearly all patterns will evolve into pseudo-random or chaotic patterns. Any structures that appear within these patterns are likely to be destroyed by the surrounding noise. Any changes occurring in local patterns will usually spread throughout the simulation space indefinitely.

Class 4: Nearly all patterns evolve into complex structures which react in complex and interesting ways, local structures tend to form within the simulation and can survive for long periods of time. Class 2 structures can potentially form however the time taken in order to arrive at the point where this begins to occur is extremely large even when starting with a simple pattern. Miniscule changes to the initial pattern will spread indefinitely. It has been theorized by Wolfram himself that some if not all class 4 cellular automata are capable of universal computation, this has been proven for rule 110 and for Conway's Game of Life.

Cellular automata are said to be reversible if for every current configuration there exists one past configuration, due to the nature of cellular automata we can think of them as functions which apply configurations to existing ones to produce a new configuration, since this is the case we can think of cellular automata as bijective (exactly pairs between multiple sets). Also not that if a cellular automata is reversible with respect to time we can consider its reversed state to also be a cellular automata.

6.5 Elementary Cellular Automata

Rules are applied to generation (n) of the simulation and determine the state of each cell for the next generation (n+1). Although there are 256 different possible rules in elementary cellular automata many of them tend to be somewhat similar to each other up to a simple transformation in their underlying geometric patterns i.e. many of these rules produce similar patterns which are simply orientated differently. The first of these transformations is called the mirrored rule which is a reflection through a vertical axis applied to a given rule which is computationally equivalent to its mirrored form. An example of this is rule 110 being reflected through a vertical line to give you back rule 124. Rules which are the same as their mirrored rule are referred to as amphirical, out of the 256 elementary cellular automata 64 of them are amphirical. The second transformation is referred to as the complimentary rule in which we can flip the vales of ever 0 and 1, an example of this can be seen when we apply this transformation to rule 110 and get an output equivalent to 137. Of the 256 automata there are 16 rules which are the same as their complementary rules. The two transformations can be combined to obtain the mirrored complementary rule, there are 8 rules which are equivalent to their mirrored complementary rules. If we apply the combined rule to rule 110 we get rule 193. Only 8 rules are the same as their mirrored complementary rules.

Two of the most prominent rules in elementary cellular automata are rule 30 and 110, these will be discussed in detail next.

Rule 110: Rule 110 is an elementary cellular automata with behaviour that borders between stability and chaos, this rule has been proven to be Turing complete meaning it can compute anything that a single taped Turing machine can, interestingly out of the 88 possible unique cellular automata rule 110 is the only one which has been proven to be Turing complete. It has been argued that rule 110 is the simplest known Turing complete system. Like the Game of Life [14], rule 110 has been shown to exhibit class 4 behaviour which is neither completely stable nor chaotic.

Rule 30: The rule 30 automata is a class 3 rule which displays chaotic, aperiodic behaviour and is believed to be the key to understanding how simple rules can in time, exhibit emergent, chaotic behaviour in nature. This rule is incredibly interesting as it produces complex chaotic behaviour from simple well-defined sets of rules. Proof of its applications in nature have been found with the appearance of rule 30 on the shell of a cone snail species known as Conus textile [22]. Rule 30 has also been used in random number generators and has been proposed as a possible stream cypher to be used in cryptography. Rule 30 also has a mirror image, complement and mirror complement in rules 86, 135 and 149 respectively. It has been scrutinized under rigorous definitions in order to be classed as exhibiting chaotic behaviour in that, it is extremely sensitive to changes in its initial conditions and it is mixing (for any two finite patterns of cells there is a configuration containing one pattern that eventually leads to the other).

6.6 Cellular Automata in 2 Dimensions

2 dimensional cellular automata exhibit some of the same traits as their 1 dimensional counterparts. There exists two fundamental types of neighbourhoods for 2 dimensional cellular automata, these are the von Neumann neighbourhood used by Jon Von Neumann in his construction of a self-replicating cellular automata. This neighbourhood consists of 5 cells (if the central cell is counted otherwise it is 4). The second is the Moore neighbourhood which consists of 8 cells (9 if the central cell is counted).

In a 2 dimensional simulation the entire grid of cells is updated simultaneously using simple rules which determine the state of the next iteration (or generation) of cells.

2 dimensional cellular automata exhibit some of the same traits as their 1 dimensional counterparts. There exists two fundamental types of neighbourhoods for 2 dimensional cellular automata, these are the von Neumann neighbourhood used by Jon Von Neumann in his construction of a self-replicating cellular automata. This neighbourhood consists of 5 cells (if the central cell is counted otherwise it is 4). The second is the Moore neighbourhood which consists of 8 cells (9 if the central cell is counted).

In a 2 dimensional simulation the entire grid of cells is updated simultaneously using simple rules which determine the state of the next iteration (or generation) of cells. In more advanced simulation software such as the Game of Life the grid is treated as a plane of infinite space allowing the simulation to progress indefinitely (constrained by processing power).

6.6.1 Conway’s Game of Life

The most famous example of a 2 dimensional cellular automata is John Conway’s Game of Life which was developed in the 1970’s. This program was responsible for revitalising interest in the field of cellular automata. Despite the level of complexity that the program exhibits the implementation of its rule set turned out to be exceedingly simple, these rules are as follows:

Each cell in the program has an alive or dead state (1 or 0 respectively) and uses the 8 cell Moore neighbourhood to determine the state of the next generation of its central cell. The two rules for updating each cell are as follows,

1. A currently dead cell will be reborn in the next generation if exactly 3 of its neighbours are alive in the current generation.
2. A living cell remains alive in the next generation only if 2 or 3 of its neighbours are alive otherwise it dies.

In basic terms rule 2 states that a cell will die of loneliness if only 1 of its neighbours is alive, alternatively it will die from overcrowding if more than 3 of its neighbours are alive.

Rule 1 dictates that a new cell will be born from 3 living cells in a previous generation.

A good description of the Game of Life can be found in Joel L. Schiff’s book “Cellular Automata: A Discrete View of the World” [14] in which he describes it as follows, “Of course this is not really a game that you play in the conventional sense but rather a microcosm of another universe that one can explore since we know its physics entirely.”

The Game of Life is an example of a highly complex class 4 cellular automata and also an example of an outer totalistic rule in that the state of the central cell in the current generation is based on the state of the same cell in the previous generation as well as the sum of its 8 neighbouring cells.

The Game of Life has also been put forward as a candidate as another cellular automata capable of universal computation, in a proof outline presented by Conway in 1982 it was theorized that using the Glider Gun automata one can transmit data from one place to another in the grid. These glider pulses can also be thought of as the electrical pulses which allow computers to function.

7 Research Approach

The following chapter will discuss the approach taken in the initial stages of the project to study how other papers dealing with practical cellular automata applications went about breaking down their respective problems into smaller, more manageable portions and then applied them to cellular automata space to produce a solution.

7.1 Initial Research Phases

Simulating flocking behaviour with 2 dimensional cellular automata has not been attempted before, if it has, it has not been published. This made deciding how to approach the problem more difficult as there is no road map or existing work to use as a reference point. While no published flocking applications exist for cellular automata there has been some experimentation in a related field known as crowd motion. One of the papers dealing with this application was written by Hubert Ludwig Klupfel, his 2003 paper "A Cellular Automaton Model for Crowd Movement and Egress Simulation**"** was a valuable resource in finding out how best to break down and approach the problem**.** Another practical application for cellular automata is the generation of musical pieces using elementary and 2 dimensional cellular automata, a paper investigating this application titled "Cellular Automata in MIDI based Computer Music" [21] written in 2005 was another valuable source of information.

During the initial stages of research a list of relevant papers dealing with these real world applications of cellular automata was compiled. Once a selection of papers which dealt with a good variety of practical applications had been collected the approach and steps used to break down the problems in those papers into simple steps and how it was converted for the application of cellular automata was studied in depth. The most useful information found in these research papers dealt with the actual conversion of continuous features such as music and flight paths into discrete space problems for the application of cellular automata. Understanding this conversion process was the key to breaking down flocking behaviour for modelling in discrete space.

In addition to studying research papers the Boids algorithm was analysed and its 3 principles, separation, cohesion and alignment were broken down into their simplest forms in order to find out what each of them contribute to the simulation and how they can be applied in discrete form. Due to the discrete, cell based nature of movement in cellular automata it was decided that the alignment principle could be disregarded, this principle deals with making each agent steer towards the average velocity of all agents in its neighbourhood which is not needed in discrete cell space as there are no velocities taken into account. Once an outlined process was established for the new approach a new algorithm was built step by step which allowed for the simulation of basic flocking behaviour in cell space.

8 Implementation

Both the Boids algorithm and the new approach have been implemented in C# using Microsoft's XNA framework. This is due to project time constraints and also for ease of implementation and rendering of results. In order to ensure fair testing the boids approach has been left un-optimized due to there being no optimization for the new approach at present and comparisons with optimized Boids falls outside the scope of this project. Some ideas on future work involving an optimized version of Boids can be read in chapter 7.

8.1 Agent Description

The following is a description of how each agent in the simulation has been implemented, all agents are identical and all apply the exact same set if simple rules simultaneously to produce their behaviour. Each agent is implemented in the following way,

- Each agent takes up a single cell at a time
- Each agent has 2 layered hexagonal neighbourhood surrounding it containing 18 cells. If a square grid of cells is being used the same setup applies with a 24 cell neighbourhood.
- The outer layer of the neighbourhood detects obstacles and other agents before they get too close to the agent.
- Inner layer is used for calculating directional forces required for separation from nearby agents once they have breached the minimum distance barrier. This barrier can be thought of as the discrete version of the radius check between agents in the Boids algorithm.
- Each agent can move to any cell within in its own neighbourhood but no further than that.

- As an agent moves between cells its neighbourhood shifts in the same manner in order to stay in the same relative position.
- Each agent can see any obstacles or agents occupying any cell within their own neighbourhood.

Figure 8.1 below demonstrates what both versions of the 2 layer neighbourhood look like when implemented in square and hexagonal grids.

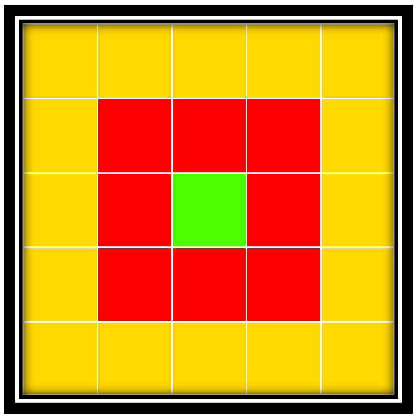


Figure 8.1 Square Grid

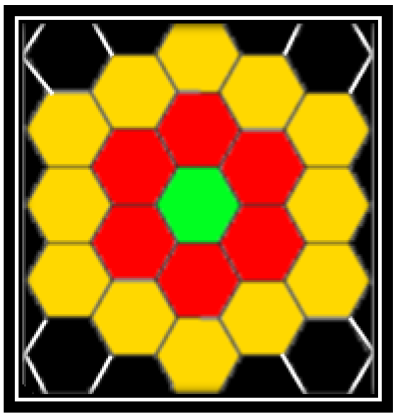

Hexagonal Grid

Figure 8.1

Agents have been implemented as simply as possible with no unnecessary attributes or information included, additional attributes such as a texture or rotational value may be required when using the algorithm in a video game for instance, this however is outside the scope of the project.

As we can see from the diagrams above, although the square cell grid offers more options for the agents next move, these moves are 45 degree turns and cause the simulation to look extremely rigid. The hexagonal approach offers less choice with only 18 cells than the square grid which offers 24 cells, it does however produce a smoother resulting path overall depending on the size of the cell grid. The lower amount of cells in the hexagonal neighbourhood also results in less checks and a more slightly efficient algorithm overall, this performance boost is discussed in the experiments section of the chapter on results and conclusions.

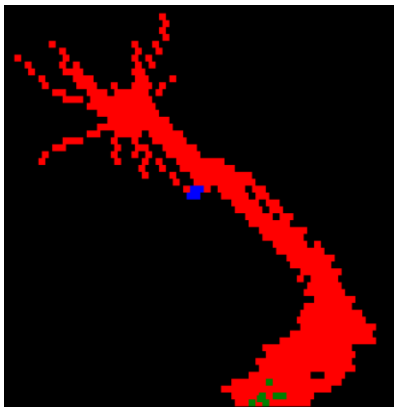
8.2 Initial Implementation Problems

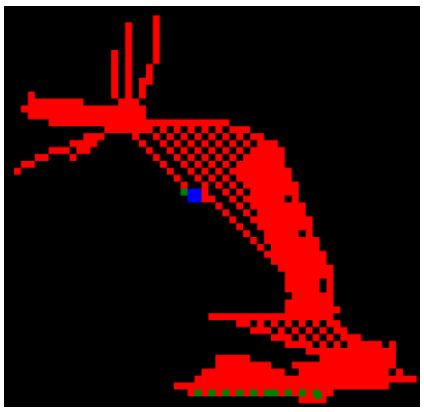
The development of this project took place in development cycles, some initial approaches showed early success but were later dropped or modified rather out of necessity or because a more efficient approach was found. These are listed below with the reasons for their exclusion from the final project implementation.

- Initially the algorithm was using pheromone trails, inspired by ant behaviours in order to relay directional information between agents. While this showed early success it became too unwieldy to implement, this was due to the need for data structures of pheromone objects which needed to be added and removed from the cell grid constantly. This noticeably reduced efficiency and introduced several bugs in the code which would have required fundamental changes to the algorithm to fix. A key feature of cellular automata is that while overall behaviour can be massively complex it is all the result of a simple rule set applied on a large scale to a sizable population of agents. With the pheromone solution becoming more complex and inefficient it was decided that a newer, less complicated approach must be found in order for the algorithm to work.

- Another early issue stemmed from leaving out a key factor in the nature of flocking behaviour. This is the knowledge that when flocks of birds and fish migrate due to seasonal changes they know where their destination lies, usually a warmer climate or a more plentiful feeding ground. While the immediate flock structure may change continuously die to internal forces the flock as a whole is constantly moving towards its destination. This factor was overlooked in the earlier pheromone based implementation, the result of which was that a flock could not be made to head toward a destination such as a food source or a safe area which of course in nature is absolutely essential for the survival of the members of the flock. The solution to this problem was to modify the algorithm to take into account certain external forces, this discussed in the algorithm section below and became a key step in getting the final algorithm to work properly.

- Initially the cell grid which the algorithm was running in was implemented as a large square grid made up of smaller square cells, this allowed the testing of early versions of the algorithm; however the resulting flock movements were far too rigid to be considered an acceptable approximation of true flocking. In order to resolve this problem the grid was re-implemented as a hexagonal grid of cells which provides each agent with a smaller number of cells to move to each step but results in a smoother path overall sue to there being more diagonal movement due to the new layout of the grid. A comparison of square versus hexagonal grid movement can be seen below,





Hexagonal Grid Square Grid

Figure 8.2

As we can see in the above screenshots the hexagonal grid allows for smoother overall turns in the flock while the square grid implementation results in several spaced out diagonal lines. Another interesting note is that collision avoidance is flawed in the square grid implementation when using this algorithm, we can see that an agent has become trapped behind the blue obstacles in the middle of the screen. This problem is eliminated in the hexagonal implementation as the more diagonal based neighbourhoods allow agents to work their way around obstacles in such situations.

- Another major problem was the size of each agents' neighbourhoods, single layer neighbourhoods are too small as agents cannot detect obstacles and fellow agents until they are already too close, this results in immediate repulsion forces being exerted causing abrupt turning and less realistic movement. It also prevents the flock from maintaining its structure and stability which is why the decision was made to extend the agents' neighbourhoods to 2 layers which results in a much more cohesive, realistic flock. Diagrams of the initial neighbourhood and the final neighbourhood can be seen below in figure 8.3,

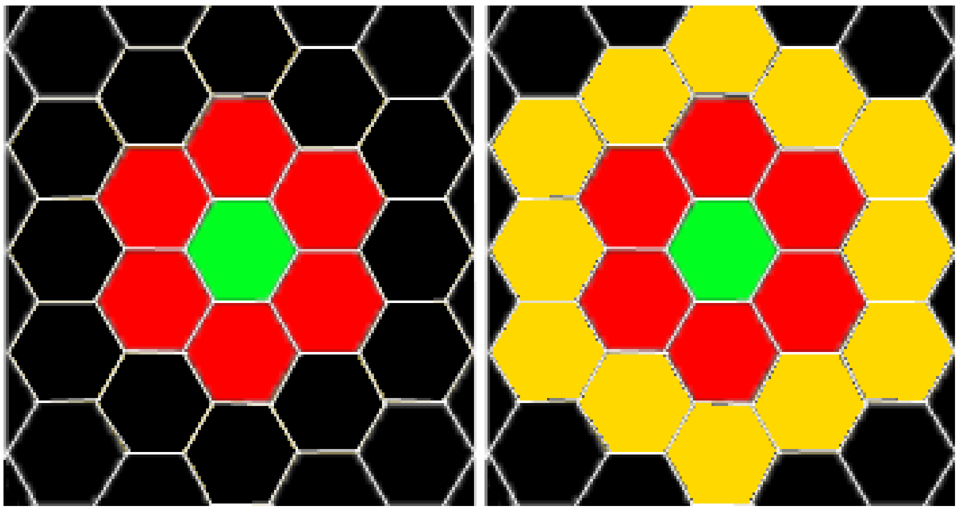


Figure 8.3

As we can see in the above diagram the neighbourhood on the left hand side was simply too small to effectively detect incoming objects. The final neighbourhood on the right detects obstacles and other agents as they populate the yellow outer cells and this allows the agent in question to take their position and direction into account in order to both avoid the agent and steer towards the centre of the local groups mass i.e. its average position without the need for last minute evasive manoeuvres.

The problems explained above hindered progress in the beginning of the project but by experimenting and investigating other options these problems were overcome and the final version of the algorithm began to take shape.

8.3 Algorithm

As explained in the previous section on implementation problems the final version of the algorithm began to take shape when a previously over looked fact was taken into account, the fact is that real flocks know where their final destination lies. This is seen in nature when birds and fish migrate vast distances as seasons change. With this factor taken into account the way the previous version of the algorithm was implemented needed to be changed. As the flocks end goal is known it eliminates the need for pheromone trails to keep flocks together and moving in the same direction.

The new algorithm takes into account distances between neighbouring agents and the directional forces which need to be applied to keep them separated but at the same time steering towards the average position i.e. the centre of mass of the agents in the immediate vicinity. This allows us to keep a cohesive flock structure and at the same time we can incorporate the external directional force vector needed for the flock to move towards its destination without destroying its internal structure.

This new ability to take into account external forces and incorporate them into internal flock forces is the key to this algorithm performing properly. A diagram of the final implementation of the basic algorithm can be seen below,

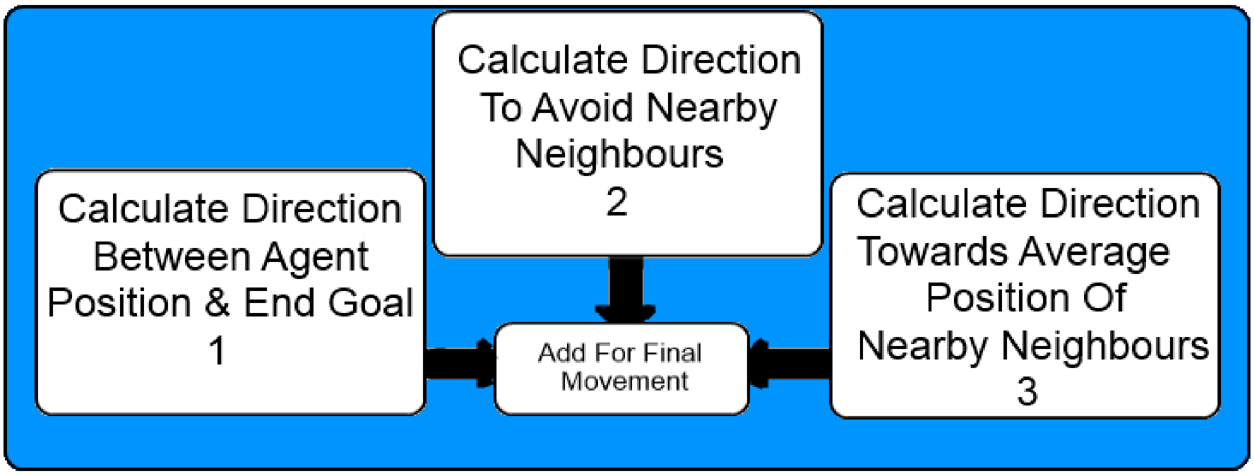

Calculate Direction To Avoid Nearby Neighbours 2
Calculate Direction Between Agent Position & End Goal 1
Add For Final Movement
Calculate Direction Towards Average Position Of Nearby Neighbours 3

Figure 8.4

The above diagram demonstrates the movement of one agent over the course of one step in the simulation, these 4 steps are applied simultaneously to every agent in the program ensuring each agent has the latest information available.

The steps are as follows,

- 1. Each agent will calculate the directional force vector required to steer towards the flocks' destination if there is one. If there is not a destination we simply do not include a vector for this step in the final calculation.
- 2. The second vector each agent needs to calculate is the directional force needed to steer away from any agents within their respective neighbourhoods. As in the previous step if a vector is not required for this step we simply do not take this force into account.
- 3. The third and final piece of information we need to calculate is the directional force required to steer the current agent towards the centre of mass i.e. the average position vector of all agents within its neighbourhood. This step can be

thought of as a discrete version of the cohesion rule as seen in the Boids algorithm.

- 4. The final stage is to add all of the forces from the first 3 steps together and get the average. Damping weights are applied to all three forces which scale based on the urgency with which they need to be applied, this allows the agent to take certain factors such as obstacle avoidance into account over agent avoidance if it is too close to a static obstacle. The resulting behaviour allows agents to force each other away from static obstacles which cannot be moved. We finally find the angle needed for the agent to rotate towards the final direction vector we get from the 3 previous steps and by treating the agents neighbourhood as a circle i.e. 360 degrees we can decide which quadrant or cell to move into next.

The diagram below shows the four steps which make up the basic algorithm, the directional vectors calculated during each step are denoted by the white arrows, the yellow square represents the flocks' destination, the green circle represents the current agent and the blue circle denotes an agent within its neighbourhood.

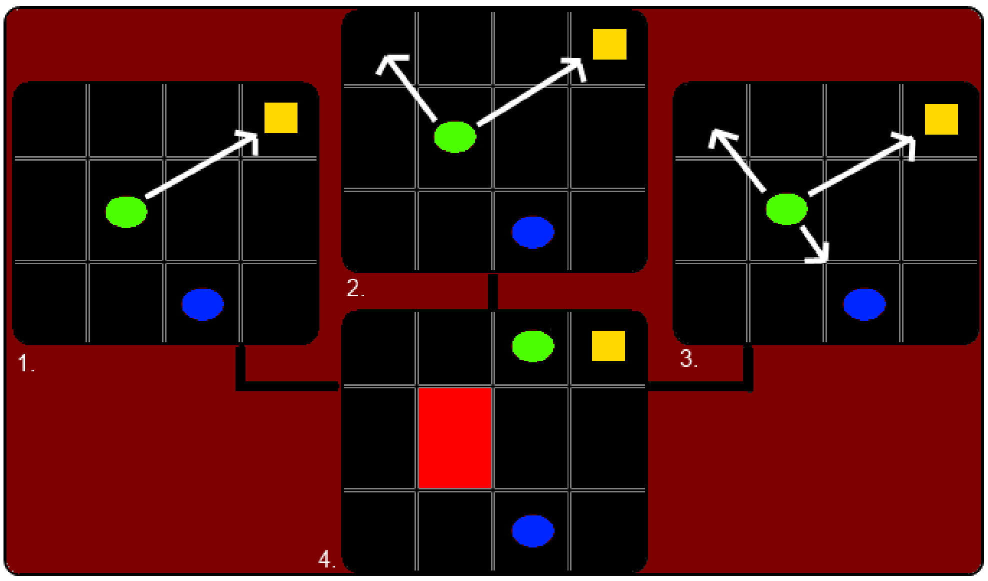


Figure 8.5

As we can observe from figure 8.5 the agent calculates the direction to its destination, the direction to avoid the encroaching agent, the direction to the centre average position of the 2 agents and extrapolates its next cell movement from them. As explained above we can incorporate damping weights to lessen the effect of any step in the algorithm which allows us to fine tune it as we see fit.

8.4 Collision Avoidance

Due to its continuous nature Boids collision can be implemented with superior collision avoidance than that of the cellular automata approach due to its agents having access to advanced knowledge of object positions in the world. This is because their vision is not limited to discrete neighbourhoods of cells. While collision avoidance does occur in the cellular automata approach agents will only avoid obstacles at close range, obstacles must enter the agents' neighbourhood to be located and avoided. Some possible solutions to this problem are as follows:

- Add a layer for avoidance on top for games using additional information, this would allow agents in game to move in a continuous manner using the underlying results of the cellular automata approach as a list of waypoints to follow in a non-cell based manner. This would allow less rigid movement when agents avoid obstacles at close range.
- Employ an expanding neighbourhood feature based on the agents velocity allowing it to see obstacles sooner at higher speeds [9]

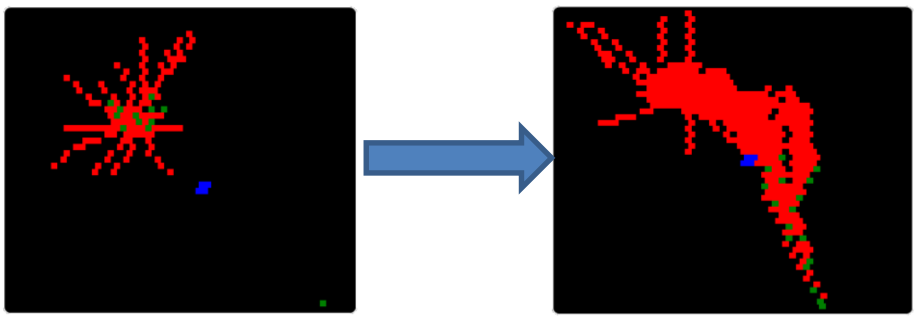


Figure 8.6

Below we can see a screeshot of Boids collision avoidance without any advanced imlementation,

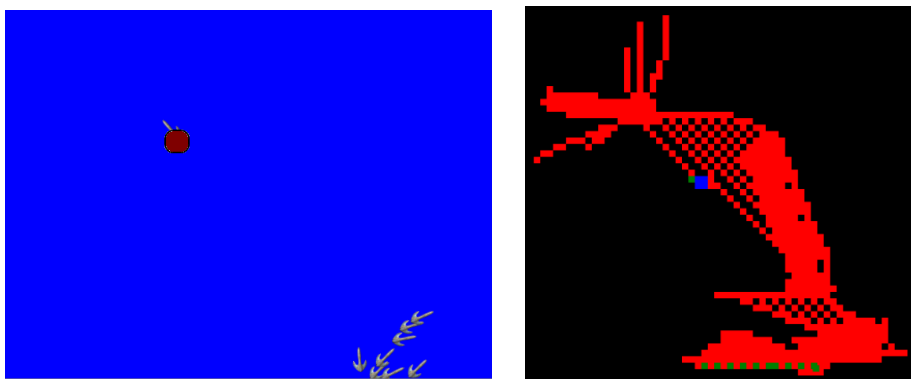


Figure 8.7

An interesting observation is that the collision avoidance in Boids seems to have the same flaw as collision avoidance in the square cell implementation of the new approach. In both instances as seen above an agent has been caught behind an obstacle. The agent can be seen on the left getting stuck under the square obstacle and on the right the agent can be identified by the green cell to the left of the blue bstacles. This flaw presents itself in both approaches when the flock is given similar start and destination points and the obstacle is in the same location relative to the agents.

It is also important to remember that while improvements are possible it would reduce the efficiency of the basic algorithm and also that advanced collision avoidance falls outside the scope of this project.

8.5 Basic Boids Implementation

The Boids algorithm was implemented in XNA in a similar manner to the cellular automata approach, this was done in order to keep platforms consistent and the performance tests unbiased. The implementation was run in a simple 800 by 600 pixel scene with variable numbers of agents and features such as static obstacles which can be added or removed on the fly.

No advanced obstacle avoidance has been implemented in addition to the algorithm as this would affect the frame rate of the program thus skewing the results of the tests. Bin-Lattice spatial subdivision optimizations have also been left out as the cellular automata approach is completely basic and un-optimized.

Identical clamping rules have been implemented for both Boids and cellular automata, agents in the Boids simulations will bounce back off edges of the grid space due to opposing forces being applied to their velocities. The agents in the cellular automata will be treated as if they have come to close to another agent and an opposing force will be applied causing them to move a cell in the opposite direction to the edge.

A screen shot of the Boids implementation can be seen below, the image shows a moderately complex scene with 400 agents seeking to reach a destination. In this image they can be seen approaching 2 obstacles in the middle of the screen and avoiding them at close range.

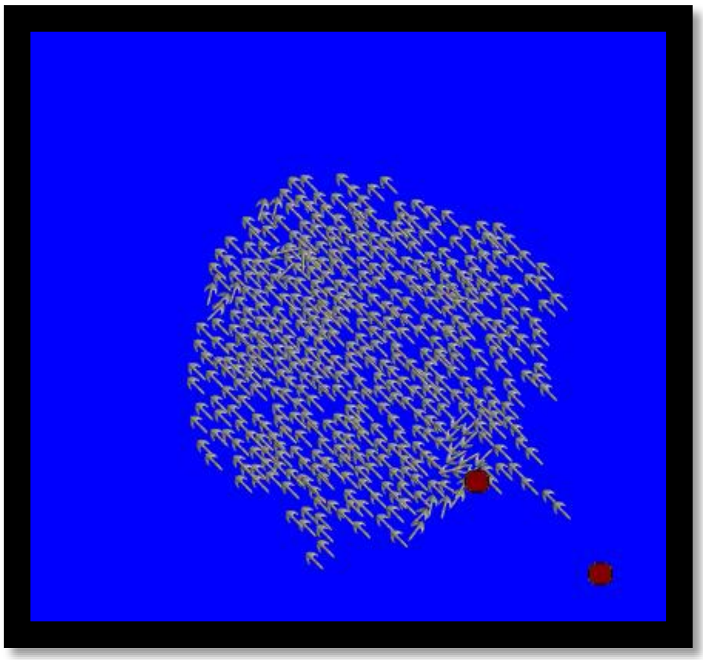


Figure 8.8

A second screenshot of the Boids implementation can be seen below, in this instance the agents are not given a final destination to fly towards. As we can see several smaller flocks have formed and begun making their way around the screen based solely on internal flock forces rebound forces from the edges of the screen space.

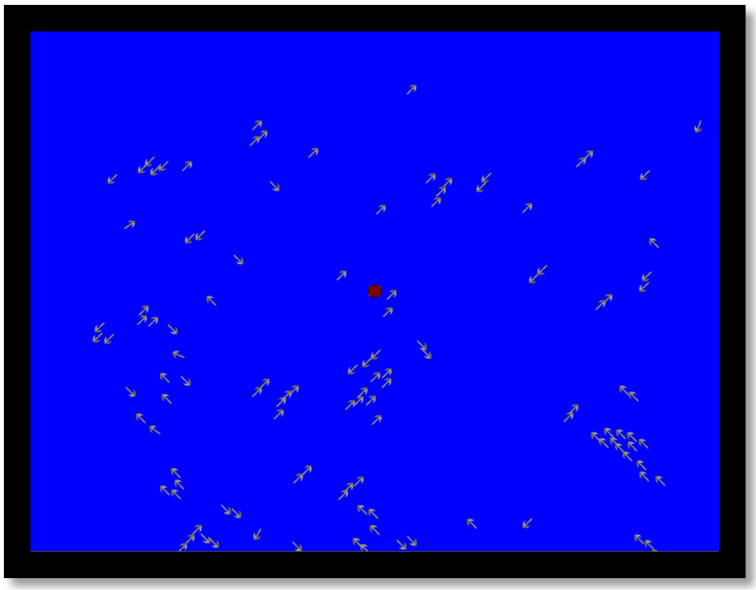


Figure 8.9

9 Results and Evaluation

The purpose of this chapter is to discuss the conditions under which the cellular automata approach was tested against the Boids algorithm. The chapter will also highlight with the aid of graphs the results of these tests and explain what this means in relation to the new approach. The aim of these tests were to identify and compare the performances of both approaches under the following criteria, frame rate, scalability under increasing complexity and algorithmic complexity.

9.1 Testing Conditions

In order to keep performance testing unbiased performance tests were run on un-optimized implementations of both algorithms. Tests were implemented using built in visual studio unit testing classes and were run separately one after another. The Boids algorithm was simulated in an 800 by 600 pixel space while the cellular automata approach used an 800 by 800 hexagonal grid of cells. For the first experiment both approaches started with 200 agents, this was increased by 200 agents at a time in a fixed time span, at intervals of 200 agents the time step between frame updates were recorded at and compiled into a line graph to easily view the results. The second experiment was used to measure the increase in complexity in the simulation as new agents were added on the fly. For the second experiment both approaches started with 100 agents, this was increased by 25 agents at a time in a fixed time span. The third test was run under the same conditions as the second with an additional object counter added in to ensure the correct number of checks were recorded. The fourth test was a straightforward comparison between the implementations of the agent neighbourhoods.

9.2 Results

Testing was performed between the cellular automata approach and an un-optimized version of the Boids algorithm.

- The new approach was found to be more efficient than this implementation of Boids under the testing criteria and conditions listed above. This cellular automata approach can simulate over 1200 agents under the provided conditions while maintaining a constant 60 frames per second which has been set as the upper limit for the purposes of this experiment.
  In contrast the Boids approach began to slow down slightly after around 200 agents, this slow down increased dramatically after approximately 450 agents and the program became unusable by the time the scene contained 800 agents.
- The new approach was also found to scale well in response to added complexity in the simulation. As additional agents are added to the world the amount of time taken to compute one step for the entire simulation increased in direct proportion to the amount of agents added. This increase in time was uniform up until the simulation began slowing down at just over 1200 agents.
  When adding agents to the Boids simulation it was found that each time a new agent was added the gap between previous and current time needed was growing exponentially thus reinforcing its O (n_2) nature. This occurs because each agent needs to search all other agents in the simulation meaning there was an increase in processing needed by every individual agent.

9.3 Experiments

9.3.1 Frame Rate

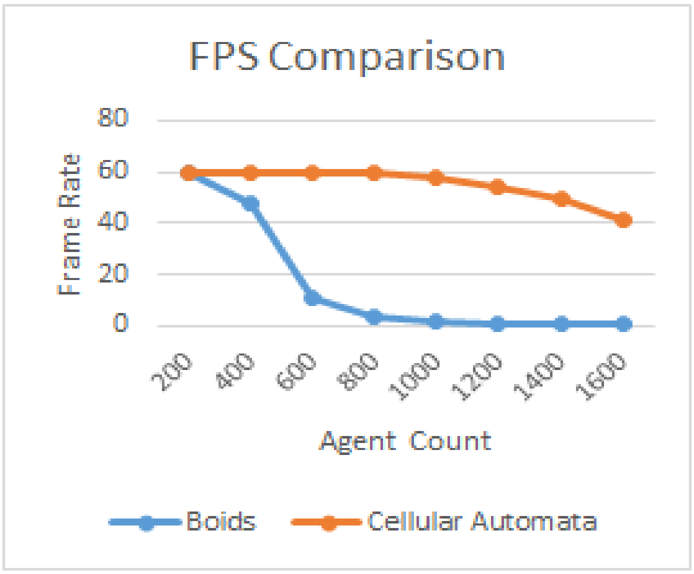


Table 9.1

By looking at the graph of results from the first experiment we can make the following observations,

- While both algorithms can run at 60 frames per second up to approximately 200 agents this implementation of the Boids algorithm reduces in efficiency quickly.
- By the time the simulation has reached 600 agents the Boids approach has dropped to a frame rate which is far too low to be used in a larger real time application.
- We can also observe that the cellular automata approach continues to run at near 60 frames per second before visibly slowing to around 40 frames per second at 1600 agents.

- While the cellular automata approach does begin to slow down it does so in a linear way with a steady decline as opposed to the Boids approach which runs fine up to a point and then dramatically decreases in efficiency over a short period of time.

9.3.2 Scalability

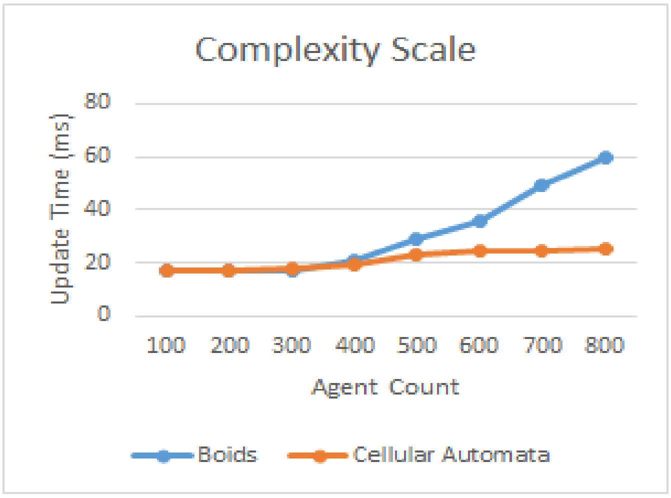


Table 9.2

The graph of results from the second experiment shows the following,

- The cellular automata approach scales linearly in proportion to the complexity of the scene.
- This implementation of Boids scales badly and in increasingly larger time steps. By the time the scene has reached 800 agents the amount of time taken to compute each frame of the simulation is approximately 60 milliseconds. The Boids algorithm is known to have a complexity of O (n $_2$) while it is un-optimized however so these numbers fall within the expected parameters.

9.3.3 Algorithmic Complexity

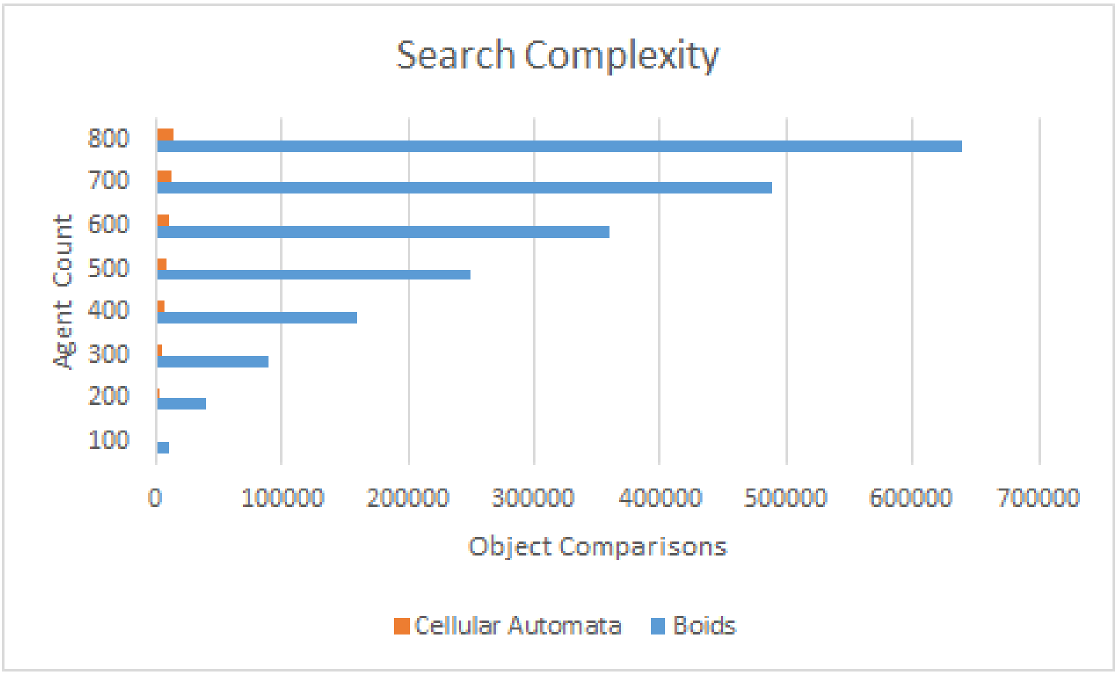


Table 9.3

The aim of the third experiment was to graph the maximum amount of objects which need to be taken into account by each approach over the course of a single frame. Both approaches started with 100 agents and this was increased by 100 until one of them became unusable. By the time it reached 800 agents the Boids algorithm was comparing agents against other agents over 600,000 times per frame. The number of checks required per frame for Boids can be expressed in the following equation,

Number of Checks = (Number of Agents * Number of Agents) – Number of Agents

For every frame every single agent must check every other agent besides itself to test if they are within their radius. Only then can an agents' position and velocity be taken into account by another.

In contrast to the Boids approach the search space for the cellular automata approach is vastly smaller and scales evenly, the equation describing the search space for a single frame of this new approach is,

Number of Checks = (Number of Agents * Neighbourhood Size) – Number of Agents

This means that each agent need only search each cell of its own neighbourhood for another agent excluding the central cell which it occupies itself. The highest number of checks the new approach needed at 800 agents was just under 15,000 for the hexagonal grid and 20,000 for the less efficient square grid which compared to the 600,000+ of the Boids approach is a massive saving in terms of time and computing power.

9.3.4 Square vs. Hexagonal Grid

The fourth experiment was a performance comparison between the two possible implementations of the cell grid. Although the hexagonal grid did prove to result in smoother movements for the flock, the efficiency gap between the two had yet to be tested. It was felt that if the square grid turned out to be more a lot more efficient than the hexagonal grid then its rigidity may have been an acceptable trade-off for extra performance.

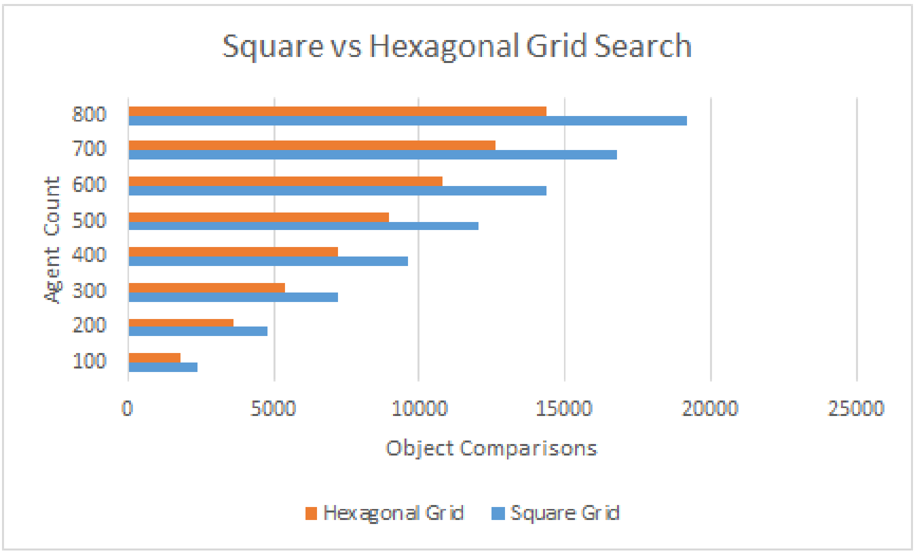


Table 9.4

This experiment was run under identical conditions to experiment 3.

As we can see from the above graph the complexity of the search increases by 25% across the board if the cell grid is implemented as a square instead of a hexagonal grid as seen in the graph below. This is due to a total neighbourhood size of 18 in the hexagonal implementation as opposed to 24 in the square one. Another advantage of the hexagonal grid is that less memory is consumed by this grid in addition to the speed increase due to the lower levels of cells that must be sorted into each agents' neighbourhoods as they traverse the grid.

9.4 Evaluation

The flock produced by the new cellular automata approach can be classified as a discrete approximation of flocking behaviour, the flock exhibits similar traits to that of a Boids flock in terms of close range collision avoidance. The flock can be guided towards an external position or allowed to roam freely around the screen space forming smaller flocks and interacting with others to form larger ones.

While obstacle avoidance does work it has less opportunity for improvement due to the fact that agents cannot see outside of their own neighbourhoods. Some ideas have been developed for how it could be improved based on ideas from other papers dealing with cellular automata applications, specifically in the area of crowd motion. These are discussed in the chapter on future work.

The cellular automata approach has proven to be much more efficient than this implementation of the Boids algorithm under the current testing conditions and it performs at a level suitable for use in real time applications.

Both the boids and cellular automata approach have been implemented in C# using the XNA framework which allowed them to be developed faster and with less implementation problems which was important due to project time constraints. It is reasonable to assume that if implemented in C++ which most triple-a standard video games are written these days, both Boids and the new approach would perform faster and be able to handle higher levels of agents. The use of pointers for assigning new cells to agent neighbourhoods and more efficient data structures could provide a massive performance boost as well as the use of low level OpenGl functions to render the agents as single coloured pixels as opposed to loading textures into XNA for every cell in the grid.

An interesting note is that due to the poorer scalability of this implementation of Boids compared to the new approach a C++ implementation could produce results which reveal a larger gap in performance between the 2 approaches.

It was found that the differences in overall curvature between the hexagonal grid and the original square cell grid is almost negligible, it is believed this is due to the smaller number of options that a given agent can move to per step. However this improvement in smoothness coupled with the increase in efficiency as discussed above means it is clearly the optimum choice for grid implementation under the current circumstances. The square grids obvious collision avoidance flaws must be taken into account as well.

10 Conclusions

The purpose of this chapter is to assess the results of this project and what has been achieved in relation to the original goals outlined in the introductory chapter.

There were several key objectives listed for this project at the start, these were as follows,

- Define and identify the key characteristics in flocking behaviour and reduce them to their simplest form in order to simplify the problem.
- Convert these basic flocking principles from a continuous to a discrete approach to simulating flocking.
- Investigate the application of the new discrete rule set to cellular automata space.
- Develop a new cellular automata that produces an acceptable approximation of flocking behaviour in discrete form similar to an existing continuous approach, in this case the Boids algorithm.
- The new approach must scale well under increasing complexity.
- The new approach must be efficient enough for use in real time applications.

Through the investigation of several academic papers based on the study of flocking behaviour, algorithms and crowd motion a fundamental outline of how a flock can be defined and how its behaviour is produced was developed. With a new concrete definition, the problem could be approached at a more fundamental level and broken down into easier to manage sections.

With the problem simplified and with existing approaches in mind it became possible to convert the principles used in continuous flocking simulation, specifically boids into a discrete form for application in cellular automata space. Previous work on crowd motion was a key resource which demonstrated how an agent can move through cellular space and how external factors such as surrounding environmental factors, threat levels and other nearby agents can influence how the agent moves.

The key step in converting these principles to discrete space was to use the results of several continuous functions such as calculating how to avoid nearby agents while still steering towards the end goal and using the results to make discrete decisions. In other words, instead of agents immediately reacting to every external stimuli such as an agent moving too close or an obstacle entering the neighbourhood it will take several factors into account and apply damping weights based on the amount of urgency involved i.e. is the obstacle still 2 cells away or will we hit it on the next step?. The results of these combined factors are used to make a single movement decision that optimally satisfies all needs.

An important stage in the implementation of these new discrete rules as a new cellular automata was the factoring in of external knowledge. While agents in the flock can only see within their own neighbourhoods they have existing knowledge of the outside world, this can be seen in nature with migratory birds. This allows us to factor in the destination as an external directional force which allows us to guide the flock as a whole while letting it handle its own internal forces.

Due to the simple nature of these rules and the lack of large data structures and sorting the new approach has tested much more efficiently than the implementation of Boids against which it was in competition. Whether it can outperform a fully optimized version of Boids using bin-lattice spatial subdivision is one possible experiment which could be done in future work on the subject.

The approach was found to scale very well during testing, the lack of sorting and absence of large data structures allows the algorithm to scale linearly in proportion to the level of agents present in the scene. The resulting algorithmic complexity of this approach is therefore found to be O (n) as opposed to the O (n_2) of the basic Boids algorithm.

Due to its linear scalability and high level of efficiency the new approach is much more suitable for use in real time applications than the implementation of Boids which it was tested against.

11 Future Work

11.1 Optimization

The implementation of this new approach has been shown in its most basic form, there are several possibilities for optimization, these include,

- Separating Axis Theorem: It may be possible to improve the efficiency of neighbourhood searches by using the separating axis theorem to detect when 2 or more overlap in a similar manner to how a physics engine compares 2 and 3 dimensional models during its broad phase collision detection stage.
- Implementing the algorithm in C++ and using low level OpenGl functions to render the scene should provide a noticeable performance boost over XNA which has been used out of necessity due to time constraints.

11.2 Multi-Cell Obstacles and Agents

Thus far the approach only allows for the placement of single cell agents and obstacles, larger objects such as mountains and walls which would take up multiple cells fall outside the scope of this project. Dealing with multiple celled objects would require some modifications to the process in order for agents to deal with them.

A possible solution to dealing with these larger objects is the use of bin-lattice spatial subdivision in the same manner it is used with the Boids flocking algorithm. As an agent approaches a larger entity the entity will be broken down into small chunks based on the agents' proximity to it. This will mean that once an agent reaches a larger entity it will have been broken into smaller sub-entities which the agent can interact with in a similar manner to single celled entities.

11.3 Optimized Boids Testing

The new approach has only been tested against an un-optimized version of the Boids algorithm. Future work would include optimizing Boids using bin-lattice spatial subdivision and performing more tests between the approaches. The optimized version would need to be tested against both an un-optimized and optimized version of the cellular automata approach with similar criteria to that used in the existing experiments discussed in the results and evaluation chapter.

11.4 Colony Building

This new approach could be used to create a model of how bees and ants build nests, communicate and forage for food. In order to accomplish this the pheromone concept used in an earlier version of the algorithm would need to be reintroduced and implemented in a more efficient manner. As pheromones are the method by which many insects communicate and leave paths to newly found food sources it is imperative that the pheromone feature be implemented correctly.

11.5 Flight Path Simulation

Accurate modelling of flight paths is an extremely important application. Taking into account dozens or possibly hundreds of aircraft, factoring in flight schedules and external forces from weather conditions and other environmental hazards is a complex problem to model. The new algorithm could be used to develop a model to visualize and optimize flight paths for aircraft. This could be possible through the use of reinforced learning techniques in conjunction with the algorithm to optimize flight paths and reduce or eliminate altogether the possibility of a mid-air collision. The application of cellular automata to such a problem has been investigated previously in a 2005 research paper titled “Cellular Automata Modelling of En Route and Arrival Self-Spacing for Autonomous Aircrafts” [19] This paper describes a straightforward procedure by which each aircraft would broadcast its position in all directions within nearby airspace. Once another aircraft enters this radius they will exchange positional information and calculate the optimal paths to avoid each other if necessary. The results of this paper bear similarities to the collision avoidance already exhibited by this algorithm so it may be that all that is required is some test data to experiment on.

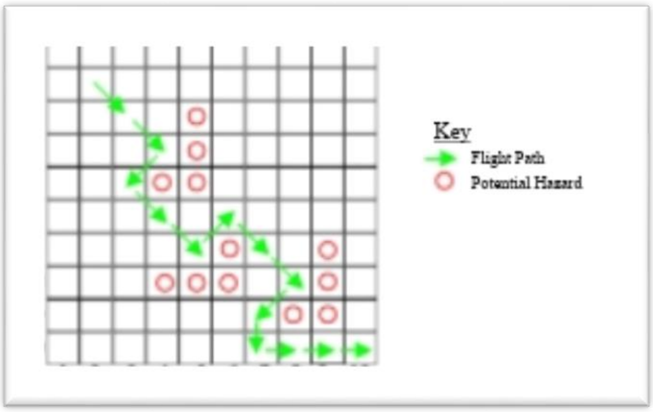


Figure 11.5

The diagram [20] above represents a single aircraft navigating a field of randomly placed no fly or hazardous zones. These zones can represent bad weather, other aircraft or solid objects such as high mountains. Figure 11.5 shows how the aircraft can detect the hazardous areas at a distance of 1 cell before taking evasive action.

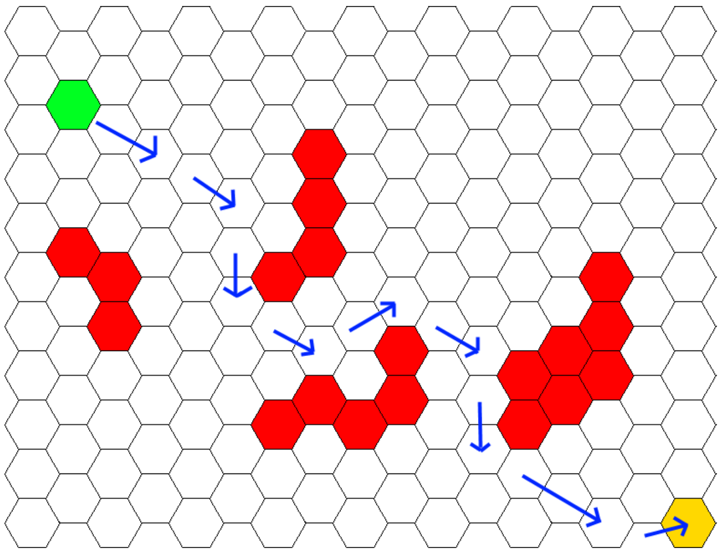


Figure 11.6

The cellular automata approach exhibits similar collision avoidance, above is a diagram of how it navigates an identical field. We can observe slight variations in the path as the agent has a 2 cell neighbourhood depth as opposed to the single layer of the previous approach. This allows the detection of hazardous area faster, additional forces such as tendency towards the final destination and multiple objects being taken into account can force the agent to move within 1 cell of a hazardous area, however the damping weights applied to each force in the algorithm will allow it to move away on the next step due to its close proximity to the object allowing obstacle avoidance to become highest priority and prevent it from colliding with the hazardous cell.

11.6 Crowd Motion

Crowd motion simulations deal with close proximity movements between autonomous agents such as fish, vehicles and even human beings, some examples of this are people getting on or off a train, vehicle movement in traffic and prey fleeing from a predator while navigating obstacles. While research has been done into this field with respect to cellular automata previously the new algorithm presented here may produce a more accurate simulation than previous attempts. A set of experiments would be fairly straightforward set up with relative ease due to the similarity of the problem space to the flocking the algorithm was designed to model. A set of scenes with randomly placed obstacles and an end goal for an agent to reach would be sufficient. A small function could then be written to record each movement direction taken to avoid an obstacle and the data could be graphed or represented visually in a diagram similar to the ones seen in the previous section on flight path simulation.

12 Bibliography

[1] Craig W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioural Model", SIGGRAPH 1987

[2] Jae Moon Lee & Hye-Kyung Cho, "A Simple Heuristic to Find Efficiently k-Nearest Neighbours in Flocking Behaviours", MATHCC 2006

[3] Craig Reynolds, "Interaction with Groups of Autonomous Characters", Research and Development Group Sony Computer Entertainment America 2000

[4] Craig Reynolds, "Not Bumping Into Things", SIGGRAPH 1988

[5] Hanan Samet, "Applications of Spatial Data Structures", 1989

[6] Zoran Popovic, "The Rapid Simulation of Proximal-Interaction Particle Systems", 1994

[7] Stephen Wolfram, "A New Kind of Science" Full book, 2002

[8] Joel L. Schiff, "Cellular Automata: A Discrete View of the World" Full book.

[9] Hubert Ludwig Klupfel, "A Cellular Automaton Model for Crowd Movement and Egress Simulation"

[10] David M. Bourg, Glenn Seemann, "AI for Game Developers, Chapter 4", Book..

[11] Craig Reynolds, "Steering Behaviours for Autonomous Characters"

[12] OJ O'Loan, M R Evans, "Alternating steady state in one-dimensional flocking"

[13] Lee Spector, Jon Klein, Chris Perry, Mark Feinstein, "Emergence of Collective Behaviour in Evolving Populations of Flying Agents"

[14] Joel L. Schiff, "Cellular Automata: A Discrete View of the World, Chapter 4.1, pg. 93 "The Game of Life""

[15] Divan Burger, Megan Duncan, Apurva Kumar, Leon van Dyk, Melissa Ingels, “HxCas Software”

[16] Martin Barksten, David Rydberg, “Extending Reynolds’ flocking model to a simulation of sheep in the presence of a predator”, KTH Computer Science and Communication

[17] The Lion King, Dir. Roger Allers and Rob Minkoff, Walt Disney Pictures, 1994. Film.

[18] Batman Returns, Dir. Tim Burton, Warner Bros, 1992, Film.

[19] Charles Kim, Khalid Abubakar and Obinna Obah, “Cellular Automata Modelling of En Route and Arrival Self-Spacing for Autonomous Aircrafts”, Department of Electrical and Computer Engineering Howard University , 2005

[20] Charles Kim, Khalid Abubakar and Obinna Obah, Cellular Automata Modelling of En Route and Arrival Self-Spacing for Autonomous Aircrafts, Department of Electrical and Computer Engineering Howard University, 2005, Image of single flight obstacle avoidance.

[21] Dave Burraston, Ernest Edmonds, Dan Livingstone and Eduardo Reck Miranda, “Cellular Automata in MIDI based Computer Music”, Creativity and Cognition Studios, Faculty of Information Technology, University of Technology, Sydney, Australia, 2005

[22] Zhenqiang Gong, Nichilos J. Matzke, Bard Ermentrout, Dawn Song, Jann E. Vendett, Montgomery Slatkin, and George Oster, “Evolution of patterns on Conus shells” , Departments of Electrical Engineering and Computer Science and Integrative Biology, University of California, Berkeley, CA 94720, Department of Mathematics, University of Pittsburgh, Pittsburgh, PA 15260 and Departments of Molecular and Cell Biology and Environmental Science, Policy and Management, University of California, Berkeley, CA 94720, 2011