An Investigation into Modelling Swarming Flies for Real-Time Simulation in an Environment with Dynamic Obstacles Using an Expanded Boids Model

James Cresswell 0801506

Abstract

This report is focused on turning the steering behaviours in the Boids model (Reynolds, 1987) from a flocking behaviour into a swarming behaviour for flies. This swarming behaviour must fulfil certain criteria which are detailed as a part of the research into this subject. Behaviour effectiveness metrics are then established based on these criteria and the behaviour is tweaked to a nominal level of perfection using these metrics.

The swarming behaviour is then adapted to also avoid dynamic obstacles of varying size and speed, and again measured up against effective metrics to be perfected. For the secondary objective, two methods of boosting efficiency are compared in order to find the most effective one for a lightweight, steering behaviour based simulation of swarming flies.

Contents

1 Introduction, Aims and Objectives

1.1 Project Aims

- To discover which industry standard techniques, if any, are useful for expanding the Boids Model.
- To use these techniques to create actors which model the swarming behaviour of flies and can avoid dynamic obstacles both effectively and efficiently.

1.2 Project Objectives

The project is broken up into two main objectives and one secondary objective. The two main objectives focus on making an effective swarming behaviour, and then adding effective dynamic obstacle avoidance behaviour to this whilst maintaining the effectiveness of the original behaviour. The secondary objective is to find the best method of making the behaviour efficient, and therefore scalable. The following list is a breakdown of the objectives:

- Adapt the Boids Model (Reynolds, 1987) to produce a swarming behaviour that works similarly to the swarming of flies.
    - Create an x86 application for Windows using pure OpenGL, written in C++ which will be the program for this investigation.
    - Define what makes a swarming behaviour similar to the swarming of flies.
    - Tweak the behaviour until it produces a visual result which fits the criteria outlined by the aforementioned definition.
    - Qualify the chosen behaviour by measuring the following:
        - Average distance from the centre of the swarm.
        - Average speed of the flies.
        - Average speed within the average distance from the centre of the swarm.
        - Average speed outside of the average distance from the swarm.
    - Measure how this swarming behaviour works over a small range of parameter variations in order to get an optimal behaviour. Parameter variations being:
        - Number of flies in swarm.
        - Minimum and Maximum speeds.
        - Distance at which flies attempt to avoid each other.
        - Weighting of separation.
        - Weighting of cohesion.
        - Maximum turning angle.
- Discover the most effective combination of algorithms and behaviours that maintain that swarming behaviour and are capable of avoiding collisions with Dynamic Obstacles.
    - Define what makes the collision avoidance a success.
    - Measure this success in effectiveness quantitatively:
        - Number of collisions with the dynamic obstacles.
    - Measure the effectiveness over a range of conditions:
        - Speed of the obstacles.

        - Size of the obstacles.
        - Number of obstacles on screen.
        - Weighting of avoidance velocity for flies.
        - Number of flies in swarm.
        - Minimum and Maximum speeds of the flies.
- Secondary Objective: Find the most efficient combination of algorithms and behaviours, whilst still maintaining the effectiveness of the above.
    - Use some basic efficiency measurements in order to see how behaviours are performing:
        - Average FPS for the game.
        - Average time spent calculating behaviour per frame.
    - Measure behaviour performance with varying parameter values:
        - Rendering.
        - Number of flies in the swarm.
        - Number of dynamic obstacles.
    - Measure effectiveness as outlined above for the primary objectives. As a result, variations on the code will be made and tested for efficiency.

1.3 Introduction

Video games have progressed a very long way since their inception. Graphics are now capable of being almost life-like in their realism, the associated peripherals and interfaces are moving into the realms of science fiction with their advances and the industry is worth multiple billions every year. However, games AI (Artificial Intelligence) is still a largely untapped area of expansion. The biggest releases tend to fall back on scripting and pre-set behaviours to make the more interesting events occur, and even on a lower level AI tends to be lacking (Fairclough et al. 2001). For example, most games will have little-to-no handling for the avoidance of dynamic obstacles. This frequently leads to situations where groups of supposedly realistic in-game actors will run blindly into physics boxes unless scripted to do otherwise, and this is a problem. Once you get movement in a group, things become worse, as they'll be knocking the boxes into each other's paths and bouncing them around, if not bumping into each other as well.

This report details a two-fold investigation into the aforementioned problem. The first part of this investigation is a research and analysis task which will determine a good base for group movement and methods to expand, accentuate and test it, whilst the second part focuses on using the results from that investigation to produce a behavioural model for swarming flies which is then to be augmented with dynamic obstacle avoidance behaviour.

The Boids Model (Reynolds, 1987) is a time-tested behavioural model developed to allow for large flocks of actors, which can be ran in real-time. It is lightweight, simple, well-documented and widely used, with the author having not only pioneered the field, but having continued to do considerable work in it with projects such as OpenSteer. This model has been used in a number of games to produce realistic, co-ordinated group movement and is generally accepted as a standard system for use in this field (Fairclough et al. 2001).

As a simple behavioural system, it is easy to integrate into games. Part of this integration is the combination with other systems and behaviours to produce the overall AI for any flocking actors. The first part of the investigation will involve researching methods that can be used to accentuate or alter the Boids Model. As this is a potentially open-ended research task, a direction to focus the research in is required.

Consequently, the second part of the investigation will drive the research of the first. Generally speaking, modern game AI is quite competent at avoiding static obstacles, such as buildings, areas that can't be navigated and even smaller obstacles such as bollards. However, dynamic obstacles still cause a problem to this day, with AI frequently walking into each other, or just bumping physics objects (such as the boxes in Half-Life 2) aside, oblivious to their presence. Whilst accepted by most players as "just how games are", this sort of behaviour can be improved. To prove as much, this project aims to have a collection of flocking actors that is capable of avoiding each other and also avoiding larger obstacles (for example, a falling box).

The one issue that using flocking behaviour would present in such a model is that the constant motion of the group isn't contained in any one area. A decent adaptation for the boids model would be swarming flies, which require a focal point but the behaviour is simpler than that of bees or ants. This decision is expanded upon in Section 2.1. With swarming behaviour, the actors (henceforth referred to as flies) being modelled will be ready to have their behaviour expanded to include the avoidance of dynamic obstacles other than their partners in the swarm.

Before moving into implementation it is important to state exactly what it is that defines swarming for flies and research into their behaviour enough to gain a reasonable understanding of why they'd be a better choice than (for example) bees or ants for this project. Section 2 covers the research phase of the investigation, which is enough to complete the first Project Aim, and also enough research to implement the first Project Objective. Only through the implementation phase can the remaining aims and objectives be completed though.

As mentioned in the sub-objectives of the Secondary Objective in Section 1.2, to make a true test of efficiency, some alternative implementations of the behaviour would be required, with alterations made to the code that are likely (or intended) to boost efficiency. Section 2.6 covers possible efficiency boosting methods, and outlines which ones are to be investigated further.

2 Research and Analysis

The first stage of this investigation is to fulfil the aim: “To discover which industry standard techniques, if any, are useful for expanding the Boids Model.” (Section 1.1) This can largely be broken down into three categories:

- The Boids Model, potential alternatives and justifying the choice.
- Potential methods, algorithms and behaviours for expanding the Boids Model.
- Selections for creating swarming behaviour based on the Boids Model.

The third of these requires a quantifiable definition of swarming that can be measured. Real-world data or sufficient models would be incredibly useful in creating such a definition, so research into this area must be done first in order to produce the definition that will focus the investigation.

Once the first part of the investigation is complete, and useful results have been collected further research into measuring effectiveness and efficiency of the swarming behaviours is required, so that a proper implementation and testing plan can be created.

2.1 Swarming

A dictionary definition of swarming is as follows:

> *“to congregate, move about or proceed in large numbers”*
> -The Free Dictionary (2011)

This definition is quite vague, providing no real parameters to work within. Swarming is simply a large number of entities moving about together. It need not even be applied to living objects. Geographically speaking, a cluster of earthquakes in an area can be referred to as a swarm, for example. Clearly a better definition is required.

From a laypersons point of view, swarming tends to be attributed to insects. Whilst this has already been proved to be a misnomer, as schools of fish and flocks of birds also qualify, amongst other groupings, it is a useful way of culling a large portion of potential behaviours to model. Continuing this line of enquiry, articles such as Miller, 2007 (writing for National Geographic) help whittle down the potential further by pointing out some of the behaviour inherent to swarming in certain types of insect.

For example, whilst bees definitely exhibit swarming behaviour, they primarily use it as a decision making process. When selecting a new nest site, bees will bivouac around the queen in a suitable safe spot (a tree branch tends to suffice) and then scouts will be sent out to find new nesting sites. They will communicate their findings with a “bee dance” to try and garner interest in their site of choice. Whoever seems most interested will slowly gather a following, and when 15 scouts all like the look of one site, they will move the nest to it.

Likewise, ants are usually seen moving in large groups, but they tend to follow each other in an orderly procession due to “hive intelligence” – a method of

reducing options which basically works on the principle that if a lot of ants are doing it, it's probably the best choice.

These two above examples are too complicated for a model to prove that dynamic obstacle avoidance is perfectly achievable, even for groups of moving actors. Locusts and flies however, tend to have a much simpler modus operandi. Locusts tend to scour an area in million-strong swarms to devour all in sight as a result of over-population. Flies on the other hand, are notable for merely gathering around objects that interest them, usually in numbers proportional to how attractive or large the object is.

A recording of a giant swarm of flies (Animal Planet, 2009), whilst not particularly scientific, does provide a decent view of what the motion of flies in a swarm is like. They cluster together, moving almost constantly. They head towards a central point or area whilst avoiding knocking into each other. Unfortunately it has proven impossible to find any scientific metrics for how flies swarm, and capturing a set of data from a real swarm of flies is beyond the scope of this project. Consequently we only have a "look" or "feel" for how swarming for flies looks, with no hard data to back this up.

To make the best of this situation, a safe option is to make an approximate model that aims to emulate the visual behaviour of the flies in the referenced video. This model can then be refined to produce an appropriate visual performance which can then be quantified as ideal values.

2.1.1 Analysis and Definition of Swarming

To conclude this section of research on swarming, a definitive definition (for this project) needs to be reached. From the material gathered, it can be concluded that flies are a suitable behavioural model to emulate. However the lack of hard data to help give parameters to the model means that data will have to be gleaned from making a visual model of flies based on the 2009 video referenced from Animal Planet.

The behaviour of swarming can be defined as such:

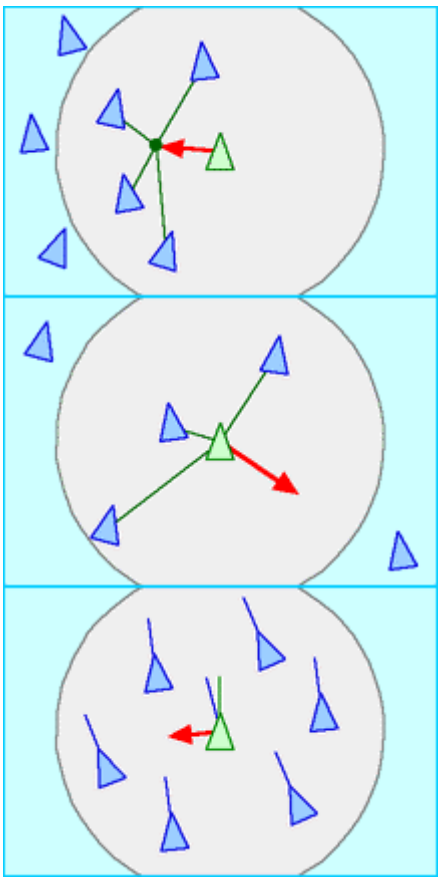
- Flies keep up constant motion, erratically looping towards a central focal point or area.
- For the purposes of this investigation, flies don't stop, hover or land.
- Flies attempt to avoid collisions with each other always.
- Flies tend not to reverse direction, but will turn almost instantly to avoid collisions.

Taking from this, the implementation will require a maximum velocity, a minimum velocity and potentially a limit on the turning angle per frame on the flies. Likewise the flies will need to cohere into a swarm around a focal point or area and avoid collisions with each other.

2.2 The Boids Model

Craig Reynolds created the Boids model in 1987, detailing a behavioural model for flocking “boids” which broke down into three different steering behaviours which he later defined (2000):

- Cohesion – “steer to move toward the average position of local flockmates”.
  - *Fig 2.2 – 1: cohesion.gif (Reynolds, 2000)*
- Separation – “steer to avoid crowding local flockmates”.
  - *Fig 2.2 – 2: separation.gif (Reynolds, 2000)*
- Alignment – “steer towards the average heading of local flockmates”.
  - *Fig 2.2 – 3: alignment.gif (Reynolds, 2000)*



The behaviour effectively and efficiently, allowed for large groups to move about in a realistic, free flowing manner, similar to that of flocking birds. Since then Reynolds has put considerable work into the field of steering behaviours, with projects such as OpenSteer and articles being put up on his website explaining how the various behaviours work and how they can be implemented. The Boids model is well documented and explained, having been around for so long.

As a result of its relative simplicity and ease of use, the Boids model has been used in a variety of games, mostly ones where group movement is highly visible, such as Real Time Strategies (Fairclough, 2001). As a lightweight steering behaviour, it lends itself well to integration with the more processor intensive functionality found in modern games.

Consequently the Boids model makes a good starting point for developing behaviour that is similar to flocking for use in games. As the industry backs this model considerably, any alternatives tend to be dismissed in favour of merely enhancing an already solid idea. However in the interest of balance, any further behaviour and movement AI techniques discussed in this section will be weighed against the Boids model for suitability as a base for this project.

2.3 Potential Models, Behaviours and Algorithms

There are a wide variety of behavioural models and algorithms available that can be integrated for use with the Boids model. This section is an examination of a variety of them in order to see what the best approach would be for using these methods to make convincing swarming behaviour that fits the criteria outlined in Section 2.1.1. As the focus is on movement AI, there are two main categories here – the first is for AI and behaviour that uses information in the environment to influence movement or navigation and the second is AI and behaviour that contains the AI/behaviour calculations within the agent.

2.3.1 Environment Navigation-Based AI

The A* Algorithm is a path-finding AI which also sees wide use in RTS games (Preuss et. al, 2008) amongst other genres. For this algorithm the game world is broken up into a square grid, and movement between squares is assigned a certain cost. The lowest cost route is then calculated from this grid. This can be updated on a step-by-step basis if necessary, and squares can be made to cost more or less to signify that they are easier or harder to traverse (Lester, 2005).

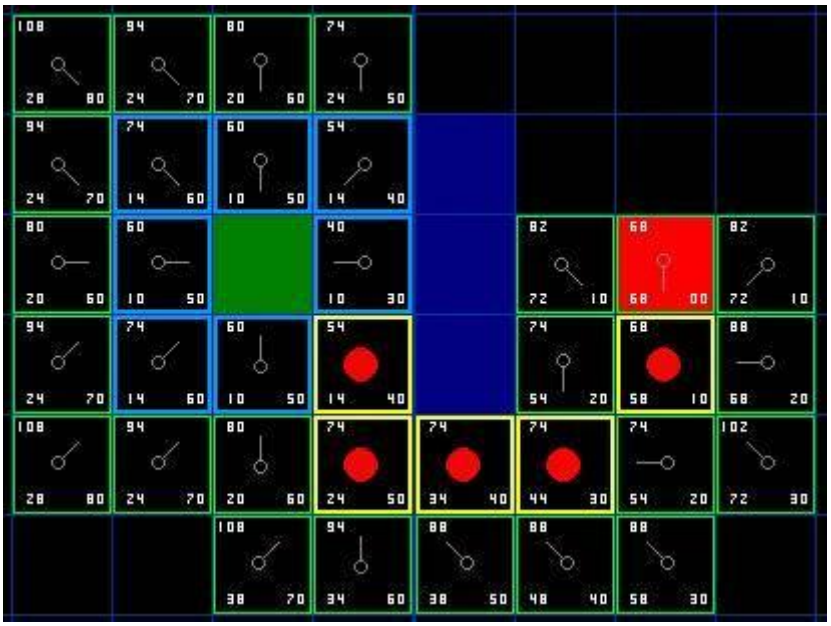


Fig 2.3.1 – 1: An example of the final result of an A Algorithm path calculation from the green square to the red square. The calculated path is the red dots (Lester, 2005).*

While this is a well-known and well-documented algorithm, which is very suitable for path finding AI, it works on a higher level than steering behaviours. Consequently it could be used with the Boids model, but ultimately isn't particularly suitable for use as a base for this project. Likewise, it wouldn't be ideal for swarming behaviour, as it is concerned more with world-traversal than how the movement is actually attained.

For an obstacle avoiding behaviour, the A* algorithm would work well for saying which area an obstacle is in, and consequently suggesting that it is to be avoided (v. d. Sterren, 2002). However it would not work well for the actual

dodging of the obstacle, so it can't serve much of a purpose in this investigation.

Falling into a similar category to the A* Algorithm, Influence Mapping also divides the game world up into a grid. Matt Buckland says this of Influence Maps in *Programming Game AI by Example* (2005):

"The term influence map refers to using a simple array of data, where each element represents data about a specific world position. IMs are usually conceptually thought of as a 2D grid overlaid on the world."

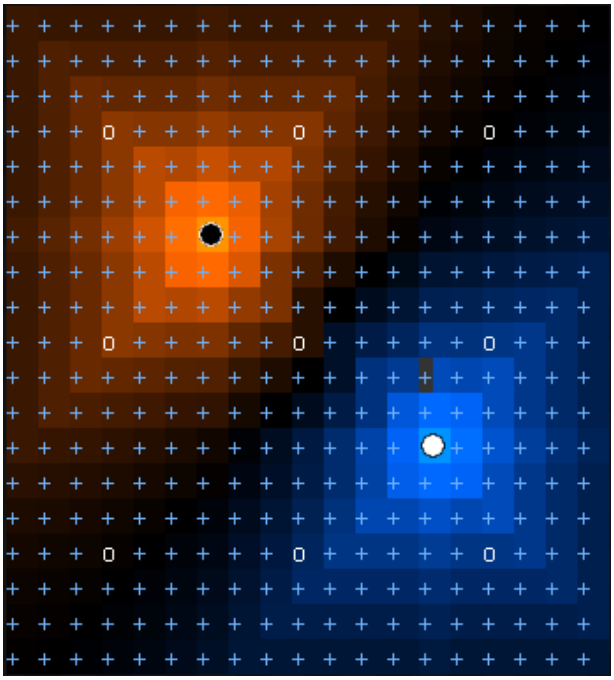


Fig 2.3.1 – 1: A graphical representation of an Influence Map showing two opposing forces (the black dot and the white dot) influencing the area around themselves (yellow and blue respectively), complete with influence dissipation. Note how the forces cancel each other out when they meet in the middle (Hansson, 2010).

Research shows that Influence Maps can be a solid alternative to the standard Boids model approach to feeding the steering behaviour with the data about nearby Boids that are to be flocked with (Flensback, 2007). However, it is also pointed out that the Influence Maps, when calculated in 3D, become quite processor intensive, particularly for larger flocks.

As a result, combining the Boids model with Influence Maps is certainly possible, but perhaps a largely pointless move, as it has more performance overhead and ultimately has little impact on the functionality. A key advantage would be that using Influence Maps would allow for the dynamic obstacle avoidance to be more easily integrated into the standing system. This is nullified by the aforementioned performance issue however, and it would add a layer of complexity to this investigation that need not be dealt with.

Flow Fields work in a similar fashion to Influence Maps. The key difference is that Flow Fields only really provide information on suggested movements. These tend to be built into game world as a static layer of information, and are usually for making movement seem natural, particularly where obstacles would lead to visually unappealing behaviour normally.

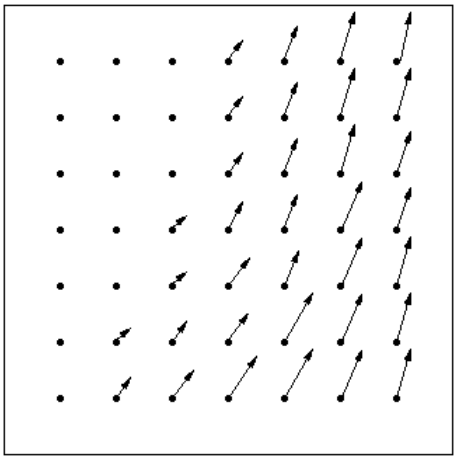


Fig 2.3.1 – 2: A simple example of a Flow Field. The length of the arrows represents the magnitude of the vector, and the general direction trends up towards the top right corner (Üngör, 2001).

In terms of logic a static Flow Field is very processor light – every entity will check the flow of the area it is in, and then be steered by it (Alexander, 2006). This could mean speeding up, slowing down, changing direction or merely staying on course. Theoretically a compact flow field could be made to simulate the movement of a swarm of flies in combination with a separation steering behaviour. However this would lead to a fairly stagnant movement pattern. Making a Flow Field dynamic would require a lot more processing, and would still only perform the same behaviour as the Boids Model at best.

Factoring in the need to work with dynamic obstacles, the main use for Flow Fields is rendered fairly useless for the purposes of this investigation. The implementation of Flow Fields would work a lot better in a game situation where it only needs to handle influencing movement around static obstacles. It is worth noting that it would certainly combine with flocking behaviours on a more general basis however.

2.3.2 Agent AI and Behaviour Methods

A time-tested method of handling AI which has to do multiple tasks are Finite State Machines (FSMs), which can be summarised thusly:

> *"A finite state machine is a device, or a model of a device, which has a finite number of states it can be in at any given time and can operate on input to either make transitions from one state to another or to cause an output or action to take place. A finite state machine can only be in one state at any moment in time."*
>
> -Buckland, 2005

As a method of making AI, they have a few distinct advantages. Namely, they are quick and easy to produce, have very little processing intensity where decision making is concerned and are well documented. They also combine well with other methods, which makes them quite versatile (Brownlee, 2002). Consequently, they can be used with the Boids model to make decisions on when to flock and when to do something else (on a higher level of AI again, like the A* Algorithm). However, the steering behaviour itself could be broken up internally using a small FSM (cohere, separate or align, for example). This could run the risk of making the Boids constantly fluctuate between two states, thus ruining the behaviour. However, implemented well this could reduce the number of calculations per frame, making this worth considering.

By applying fuzzy logic to FSMs, we can make them “Fuzzy State Machines” (FuSMs) infinitely more flexible. Essentially a state would stop being either “yes” or “no” (binary logic) and become “somewhere between 0 and 1” (fuzzy logic). So, for example, something could be 25% hungry and 75% thirsty, meaning it might consider food, but is mostly interested in having a drink. The way to make this work is to use thresholds and scalars to determine which states are active. The implication here is that FuSMs can be in more than one state at a time. (Cruz, 2008)

Logically these provide a lot more opportunities for mistakes and errors to arise, but are quite powerful. Obviously needing to work out which states should be active and by how much requires more processing and more logic calculation. However, the results are a lot more flexible, so the trade-off between FuSMs and FSMs can be considered reasonable.

Realistically, this could easily be mixed with flocking behaviour to provide a dynamic weighting system to the various steering forces at work, based on the various parameters that go into deciding the steering forces (proximity of other boids, etc.) This could, however, prove to be a pointless over complication, as a lot of the strength in the Boids lies in its simplicity and effectiveness as-is. In much the same way that FSMs could be used, FuSMs can also be used, but with a greater risk to the efficiency of the behaviour calculation.

The Boids Model is comprised entirely of steering behaviour, which works on a lower level than higher-level AI such as FSMs and FuSMs (Reynolds, 1999). Consequently there is a strong case for working at this level with any expansions or alterations to the behaviour. There a wide variety of applications of steering behaviours in regards to desired locations, obstacle avoidance, entity avoidance and entity chasing (Reynolds, 1999). Simply put, altering the steering behaviour is very possible, and is most in line with the basic premise of this investigation.

As a swarm of flies must not have any semblance of organisation in its movement, removing the alignment part of the flocking behaviour in the Boids model would be a good start to modelling this behaviour. Tweaks to the way cohesion and separation work afterwards can produce a decent steering behaviour which can be further altered until it resembles swarming.

The dynamic obstacle avoidance is partially already implemented in the separation behaviour, as the other Boids are technically dynamic obstacles for the Boids in the flock. Simply making the entities aware of dynamic obstacles (and their size, in better simulations) and having an added behaviour that provides their movement with an added "repulsion vector" away from the obstacle is enough to smoothly avoid obstacles whilst heading towards a goal (Johnson, 2004).

Continuing on the level of steering behaviour, Inverse Steering Behaviours are a good way of limiting the results of steering behaviours to get repeatable results. A discrete set of points is projected around the entity, which give it a number of options for movement (sidestep, step forwards, etc.). The steering behaviours are then used to calculate a value, per point, between 0 and 1 based on how close the behaviour would get to each point. Whichever point gets the highest aggregate value is then moved towards by the entity in question (Thunderhead Engineering, 2009).

For individual agents and small groups of agents, this is a great way of making the outcomes from a steering behaviour at a given time into an easy to alter number of predictable options. Performance-wise the additional processing required is relatively minimal and this method works on the same level as the Boids model, therefore making it an obvious candidate for use. However, as larger numbers build up the additional processing does become an issue.

"Fortunately, the precision of movements is less important if you steer large quantities of agents; for example a flock of birds. As long as the flock behaves naturally, it does not matter if a single agent performs suboptimally, so classical steering behaviours are the weapon of choice in such a situation."
- Amor et al. 2006, p. 223

As pointed out in the above quote from the article "3.5 Fast, Neat, and Under Control: Arbitrating Between Steering Behaviours", this amount of control and perfection on a per-agent basis would be lost in a suitably sized flock or swarm, and therefore is unnecessary for this investigation.

2.3.3 Analysis and Conclusion from this Section

Looking through the previous two sub-subsections a fairly convincing argument can quickly be made for the use of altering the Boids model at a behavioural level and then adding to and changing the steering behaviours to best produce a convincing model for swarming flies.

As this would keep the initial processing requirements very similar to the Boids model there is a good point of reference when measuring efficiency of any additional methods and behaviours. In turn, working with steering behaviours requires no grids of data to be generated, unlike with the A* Algorithm, Influence Mapping and Flow Fields. Likewise there are no large overhauls of structure to the AI in the flies to contend with, which ultimately eases the transition from Boids and flocking to flies and swarming.

The conclusion to be drawn here is that, whilst all of the methods, behaviours and algorithms investigated here are certainly possible to combine with the Boids model, altering the steering behaviour is most appropriate for this project.

2.4 Potential Starting Points

As has been previously mentioned, Craig Reynolds has put considerable work into the field of Steering Behaviour, and many others have joined in since the ideas inception. Now, roughly 24 years after the original paper detailing the Boids model was written, there are an abundance of websites detailing how to make a Boids model quickly and easily, with various academic articles in books such as the *AI Game Programming Wisdom* series being dedicated to working with steering behaviours in games.

This section compares starting points for the implementation phase of this investigation. Obviously the better the starting point for the implementation, the quicker the results from the testing phase can be gathered. This research section also helps with an overall understanding of steering behaviours and shows a sample of the resources currently available online in this field.

2.4.1 The Boids Model

The ease of use and large amount of material available online are strengths of the Boids model; likewise, the Boids model is definitively where the roots of the implementation are seated. Therefore the obvious choice for a starting point is to make a basic Boids implementation, and then to remove the alignment part of the behaviour before then working to make it fit the criteria for swarming outlined in Section 2.1.1.

This is certainly an acceptable course of action, and sample code provided online would allow one to gain a strong understanding of how to accomplish it (Reynolds, 1997-2004). However, if a sample project were to exist which would provide a better starting point, either with a complete and easily modifiable Boids model implementation or an actual swarming implementation, using it would be a wise choice. If there are no other decent starting points however, this one would suffice.

2.4.2 OpenSteer

OpenSteer, an open source steering behaviour library and demonstration started in 2002 by Craig Reynolds and developed up until 2006 by him and other contributors, is an obvious port of call when considering working with the Boids model and steering behaviours in general. The code includes a perfectly functioning Boids model which would certainly remove the need to write the code for one before then stripping it of its alignment behaviour.

A brief e-mail conversation with Reynolds himself (fully transcribed as Appendix A) did bring to light the fact that the code is mostly for demonstration purposes, and is not to be considered perfect code for inclusion in a full game. For the purposes of this investigation, this wouldn’t cause an issue until efficiency testing is to be considered, where there may

then arise the issue of requiring to streamline the code somewhat (a problem that would be just as prevalent in any other solution).

Consequently using the code from OpenSteer makes sense, when compared to writing a new Boids model implementation from the information to be found in the documentation and other examples. It would require less work to reach the point where swarming behaviour is in existence, and the code started from would be written by somebody who really knows what they are doing.

However, there is still the possibility that there may be a swarming behaviour implementation already made. If it happens to use a combination of cohesion and separation behaviours to achieve swarming, then it would be a much better starting point, and would merely require tweaking in order to fit the parameters that define swarming behaviour. The decision here is now to be decided by whether or not there is such a behaviour already pre-made for open source use and modification that fits the criteria. If there isn't one, OpenSteer is the starting point of choice.

2.4.3 An Open Source Swarming Behaviour

There is a lot of open source steering behaviour to be had as a resource for people wanting to work in the field. A lot of it is either provided as part of papers or as small projects. Evidently a lot of the work can be attributed to Craig Reynolds in some way, as he started it all off with the Boids model. However, the one standout open source swarming behaviour as a candidate for the starting point of this investigation was uploaded to *Open Processing* in the February of 2010. This project, simply titled "Swarm" is described on the project page as:

"A simple swarming algorithm for flies that (a) cohere and (b) avoid being too close to each other"

-Turner, 2010

The code clearly uses steering behaviour as its base, and is conceptually similar to the Boids model. Whilst the simulation is designed to keep the flies within the confines of the rendering window and is written Java (rather than C++, which is the language this investigation will be using) the algorithms and overall behaviour are exactly what this project would be building on. One interesting point to note is that this implementation specifies that either flies move to the average position of every other fly in the swarm, or that they avoid their nearest neighbour (weighted by distance to that neighbour) if they are within a certain range.

2.4.4 Selecting a Starting Point

Of the three investigated starting points, all are reasonable choices for this project. Starting by writing a Boids model implementation in C++ would largely be reinventing the wheel, whilst using the OpenSteer implementation bears the minor disclaimer about code quality from the author. Of the two, starting from the OpenSteer implementation is the most logical, due to the reduced workload.

However, taking into account Swarm (Turner, 2010), which is a lot closer to the intended swarming behaviour than the Boids model is, the choice is not so clear cut. On the one hand, the OpenSteer implementation is written in C++, but on the other hand the algorithms in Swarm would reduce the amount of work required to get the swarming behaviour up and running.

On reflection, translating the algorithms in Swarm from Java to C++ would be quicker than turning any Boids model implementation into the swarming behaviour outlined in section 2.1.1. As a result, the starting point for the implementation phase will be Swarm, which is to be converted to C++, and then tweaked, altered and expanded to fulfil the criteria for this investigation.

2.5 Testing Metrics

For this investigation to be scientific, quantifiable measures must be taken in order to qualify success when it comes to making the steering behaviours effective. These measurements should conform to a definition, such as:

"Measurement is the process of empirical, objective, assignment of numbers to properties of objects or events of the real world in such a way as to describe them."

-Finkelstein, 1982, p. 6

This will ensure that the measurements taken have some scientific merit, and therefore will produce valid results. There are two different areas of testing. The primary area is “effectiveness”, a term that can’t be quantified easily, as effectiveness means different things for different people. The description used in this investigation will be “Behaves as expected and gets the best results for the numerical criteria supplied”. The secondary area involves software metrics, and will be a lightweight investigation of efficiency.

2.5.1 Metrics for Effectiveness

Section 2.1 attempted to define swarming for flies using hard data, which proved difficult to procure within the means of this investigation. Consequently an unscientific real-world video clip (Animal Planet, 2009) is being used to gauge effectiveness. This does not mean metrics can’t be employed here though. Section 2.1.1 defined what will make the swarming behaviour look correct. From that definition the key measurable behavioural metrics are:

- Collision avoidance, which features largely in both the swarming behaviour and the dynamic obstacle avoidance investigation.
    - Aiming for a low collision count between flies and the dynamic obstacles is a clear choice.
- Motion and position.
    - Average speed of the swarm.
    - Average distance of the flies from the centre of the swarm.
    - Average speed of the flies within the average distance.
    - Average speed of the flies outside of the average distance.

2.5.2 Metrics for Efficiency

Although a secondary objective, the measuring of efficiency is important, as this investigation and the results thereof can't be used in a game if the code is too inefficient to run with any current generation rendering engines. According to Kaner and Bond in their 2004 paper on Software Engineering Metrics, there are many brands of efficiency for programming. For instance, the efficiency of the actual programming itself can be measured, although that is unnecessary for this investigation.

This investigation is instead interested in the efficiency of the code at run-time. Measures such as Lines of Code and Functions Written per Day are excluded from this set of measurements and instead the intensity of processing and the memory usage of the behaviour are important. As this is a secondary objective there will only be a cursory investigation; two or three metrics will suffice.

The first, and most common, choice in metrics is Frames per Second, or FPS. If the behaviour, running within reasonable parameters, lowers the FPS below the cap of ~60 that some computers enforce, then it is definitely not efficient enough for use with highly complex and resource intensive systems. A second choice for this investigation would be more focused on the behaviour itself. There are a few options here, such as checking the big O value for the behaviour - after all, with no attempt at refining efficiency the swarming behaviour demonstrated in "Swarm" is an n^2 problem, which means that the behaviour has to calculate once for every object, for every object (or that the number of calculations is equivalent to the number of objects, squared). Alternatives to working to lower the big O value are reducing the number of function calls made to the bare working minimum, or examining how much time the behaviour takes to calculate on average and seeing where improvements in speed can be made.

Timing the behaviour calculation per frame can be achieved through recording the time taken per object and calculating an average for the frame in question. The goal here would be to reduce the time taken to the lowest possible. Whilst a third metric, such as a measure of memory usage, is also possible, Kaner and Bond do raise another point which is worth considering – the measuring of software metrics can in itself be inhibitive. The extra processing required to collect, record and present the data can actually skew the results somewhat, particularly with low intensity calculations.

To summarise, the two software engineering metrics chosen for efficiency testing are Frames per Second (FPS) and the average amount of time spent calculating the behaviour per object, per frame (in milliseconds).

2.6 Efficiency Boosting Methods

The second area of research regarding the secondary objective is ways to improve efficiency. Due to the two metrics being time-based (behaviour calculation time and frames per second) the key way to improve efficiency based on this measure is to reduce calculation time.

This can only really be achieved by reducing the number of calculations needed to be made, as methods like multi-threading vary heavily based on the machine in use and areas such as rendering efficiency aren't being tested. A key method of reducing the number of calculations is to reduce the code down to its basics. The less lines of code there are to handle, the less functionality there is that would require calculating.

Another key method for reducing the number of calculations is to bring the big O value as near to n as possible. The best way to do this is to cut out loops as much as possible, or reduce the number of iterations required at any one time. Spatial partitioning methods (such as Influence Maps) are a good way to do this, although they do have their own performance overheads, which tend to merely move the processing elsewhere. Whilst this would improve the time spent to calculate the behaviour, it would have no real effect on the FPS, and if implemented badly would actually reduce the FPS.

Using FSMs within the behaviour algorithms can be a good way of only doing the calculations required at the time, although this would have to be implemented very well so as not to compromise the effectiveness of the behaviour in any way. The real issue when working with efficiency is that every idea may well have a drawback or unforeseen consequence which would either nullify the positive effects of the implementation, or require considerable reworking and improvement in order to truly have the intended effect. By far the biggest concern is that the measures taken to improve efficiency will overcomplicate the program.

A conclusion that can be reached here is that efficiency is important, but relatively difficult to improve when it comes to the solid implementation of a simplistic idea, such as steering behaviours. The various avenues suggested in this section will all be tested to see where the best implementations are and what comparisons can be drawn.

3 Testing Plan

Section 2.5 outlines the testing criteria for this investigation, which, to summarise, are:

- Effectiveness
  - Number of collisions
  - Average speed of the swarm
  - Average distance from the centre of the swarm
  - Average speed within the average distance
  - Average speed outside of the average distance
- Efficiency
  - Average Frames per Second (FPS)
  - Average time (in milliseconds) spent calculating behaviour

These criteria cover the primary and secondary objectives of the investigation, which can be summarised as:

- To create effective swarming behaviour for flies, which conforms to the criteria listed in section 2.1.1 and is based on the Boids model by altering the steering behaviour.
- To add to this steering behaviour so that the flies can effectively avoid dynamic obstacle.
- Secondarily, to find the most efficient implementation of the above behaviours.

As these are simple metrics the tests should also be appropriately uncomplicated. There are three key stages of the investigation (each linked to an objective) and each one has certain metrics related to it.

In order to best analyse the implementation a variety of parameters that can be altered is required. These variables will be limited to a range which will need to be established during the implementation.

The actual variables (or parameters) being altered will vary from objective to objective, so the below is an overall listing of parameters that can be used, with short explanations as to why for each:

- Number of flies
  - Larger swarms will have more intense movement than smaller ones, so testing collisions and average speeds and locations will help prove the behaviour is effective for all sizes of swarm.
  - Scalability will also prove useful when measuring efficiency, as algorithms with a high big O notation value tend to only show their weaknesses as the number of objects grows, as it takes longer per object to calculate the behaviour.
- Minimum and Maximum speeds of flies
  - Only useful for testing effectiveness, particularly when it comes to avoiding collisions. Obviously the metrics involving average speed will have to be relative to these speeds if they aren't set as a constant.

- Distance at which flies attempt to avoid each other
    - Separation behaviours work within a maximum distance for best effect, particularly when combined with cohesion behaviour. As a result, the maximum distance can be a variable, meaning flies attempt to dodge earlier or later.
    - This can be used for both effectiveness and efficiency testing, if the number of calculations for separation is based on distance. If not, it is still very useful for effectiveness across all metrics.
- Weighting of separation behaviour
    - The suggested vector for the separation behaviour can be multiplied by a scalar to give it a certain amount of weight or "urgency". Fractional values would make flies seem less concerned about colliding with each other, whilst higher values would make flies avoid each other in an exaggerated fashion.
    - Only useful for tweaking the behaviour for effectiveness.
- Weighting of cohesion behaviour
    - Similar to separation, but affects how much the flies want to get into the centre of the swarm (or, in some implementations, how much they want to get in the centre of their nearest neighbours).
    - Again a useful parameter for tweaking effectiveness.
- Maximum turning angle
    - Limiting a flies turning angle would be a good method of altering effectiveness and forcing perfection from tweaking the other parameters. "Swarm" doesn't do this however, so technically the value would start as a constant.
    - As an effectiveness parameter that may not apply realistically to flies (which are quite agile), this one will only be tested if an implementation of it produces behaviour that still visually satisfies the criteria.
- Number of dynamic obstacles
    - The more dynamic obstacles there are, the more the flies equipped with the dodging behaviour will have to work to not have any collisions.
    - Useful for testing effectiveness and efficiency of the dynamic obstacle avoidance behaviour.
- Dimensions of the dynamic obstacles
    - Purely an effectiveness testing variable, the scale of the dynamic obstacles should have an impact on the behaviour, otherwise the number of collisions will rise because the flies aren't aware of what they're meant to be dodging.
- Speed of the dynamic obstacles
    - Faster obstacles will require lower reaction times of the flies, which should have a lower limit to them, meaning if something comes at them quickly enough, even a very effective behaviour should still be incapable of avoiding it. Determining an acceptable value here should be done in conjunction with the distance that flies become aware of an obstacle and the speed at which the flies can move.

- Distance that flies become aware of an obstacle
    - Has a minor impact on efficiency, but is mostly concerned with effectiveness.
    - Should be based on the dimensions of the dynamic obstacles, and should also take into account the velocity of the dynamic obstacles, as it will determine how early they move to avoid the obstacle.
- Weighting of the avoidance behaviour
    - Like with the other behaviour weightings, will be used to tweak effectiveness by making the flies move more or less to avoid collisions.
- Implementation
    - Whilst effectiveness is most concerned with finding the best parameter values for an adequate set of steering behaviours, efficiency will be more concerned with getting the behaviours to run as efficiently as possible. As a result, a range of different implementations of the steering behaviours will need to be created in order to see which one runs the quickest.
- Rendering being on or off
    - This Boolean choice is purely aimed at checking efficiency of the behaviour without any of the rendering overhead involved to sully the performance.

The first stage of the implementation is the creation of a swarm of flies, which will use “Swarm” (Turner, 2010) as a base, as explained in Section 2.4.4. As a simple, steering behaviour based swarming algorithm, “Swarm” will need some additions and alterations in order to fit the criteria for a swarm of flies, as defined in Section 2.1.1. Once these criteria are met on a subjective, visual level, the behaviour can then be measured using the effectiveness metrics over a range of parameter values. The parameters being altered will be number of flies, maximum and minimum speeds and weightings of separation and cohesion behaviour; section 5.1 has the ranges for these values. The number of collisions between flies will not be measured as the implementation is a 2D implementation of 3D space. Whilst the flies are trying to avoid the positions of other flies as a part of their separation behaviour, in a full 3D swarm they’d be packed into a tighter area than in a 2D swarm where they are all successfully avoiding collisions. This discrepancy would be very noticeable, and would have a large impact on the results of later testing stages if left unconsidered.

The second stage of the implementation is to adapt the swarming behaviour above, complete with the most effective parameter values, to include additional behaviour for avoiding dynamic obstacles. These dynamic obstacles will have variable velocity and size, and the overall aim is for the flies to be able to avoid collisions with them successfully under reasonable conditions. Effectiveness will again be the measure here, with the parameters being focused on being number of flies, number, size and speed of dynamic obstacles and the weighting of the avoidance behaviour; section 5.2 has the ranges for these variables. Maximum and minimum speeds of the flies will be assumed perfected from the previous stage of investigation, as will the

weightings of the swarming behaviour. This should then move the focus onto just the obstacle avoidance behaviour, and the only metric taken will be number of collisions between flies and obstacles.

The third and final stage of the implementation is to fulfil the secondary objective of finding the most efficient implementation of the complete steering behaviour, as perfected in the second stage. Using the efficiency metrics, the basic behaviour and a number of variants that have been programmed using the efficiency boosting methods suggested in Section 2.6 will be tested. The parameters here will be checking scalability with the number of flies and dynamic obstacles, checking pure running speeds by turning the rendering off as well getting metrics values with the rendering running. Efficiency boosting methods may also involve altering the way the behaviours work in a fundamental way, and where this happens additional testing to check efficiency of the other implementations attempts at emulating that behaviour will be considered. However, if, for example, a spatial partitioning method means that the flies only check for other flies in a certain range when cohering, this behaviour will not be emulated in other implementations as this is part of the efficiency boosting. Section 5.3 details the range of values used for any tested variables.

4 Implementation

The implementation of this project is split into three sections, with each covering one of the summarised objectives in Section 3. Overall the project will be written in C++, making use of pure OpenGL for rendering. To ease rendering, glBegin(), glVertex3f() and glEnd() will be relied on – whilst these rendering methods aren't the most efficient, they are simple, and as the rendering of the flies isn't being tested, this is a non-issue. A code base has been put together to make a "game engine" which handles rendering, updating of in-game objects, input, text output and timing.

The resources used as a base for this project are the vector, matrix, timer, random (Lager, 2006) and BFont classes, which have all been provided by the University of Bolton. They have been selected due to familiarity and ease of use. Likewise the component-based architecture, AABB collision boxes and game class are all adapted and improved from the source from *Assignment 2: The Game* (Cresswell, 2010). Movement and steering behaviour are written explicitly or adapted for this project, as are the metric and parameter management classes.

The program is designed as a 2D representation of a 3D space, with 3D rendering and the code in place so that it could be easily made to run as a full 3D application. The source code for the all implementations is available on the accompanying CD.

To help trace the behaviour of the flies, they are rendered as square-based pyramids, which point in the direction they are headed. The rendering also leaves a trail which links the last 120 points that the object was at, producing a coloured line which makes following the behaviour more visually obvious. Likewise, to help collect metrics data the game will display the relevant information as text, and allow the user to manipulate the parameters in-game to set up the testing situations quickly.

4.1 Implementing Effective Swarming Behaviour

Due to the use of steering behaviours in this investigation, a movement component was created which could either update the position of its parent object by that object's velocity, or poll the parent object's components for any steering behaviour components. This poll would result in an average suggestion which is then added to the current velocity and the update then proceeds as normal. The speed is kept between a minimum and maximum value, which is specified when the flies are created.

The original plan was to create classic Boids flocking behaviour first, in order to become familiarised with the behaviours. Once the Boids model was implemented, the alignment behaviour would be removed, and the steering and cohesion behaviours tweaked to produce a reasonably stable swarm. A new third behaviour was then introduced which focuses the swarm on the mouse cursor. The UML diagram for this implementation (featuring some unused classes) is in Appendix B as Fig 1.

However, this behaviour visibly did not fit the criteria for swarming as stated in Section 2.1.1, as the flies would always hit the centre point of the swarm almost exactly and the overall motion was more like a 2D representation of orbiting in a 3D space. Further research was undertaken, which unearthed "Swarm" (Turner, 2010), which is covered in Section 2.4.3. The behaviour present in "Swarm" was considered a much better starting point (Section 2.4.4) and consequently a C++ conversion of the Java code was created in order to replace the three current steering behaviour components. This new component was considerably closer to the mark, as it did not feature any stable orbits or sling-shotting behaviour. A final UML diagram for this implementation is in Appendix B as Fig 2.

Maximum turning angle was investigated as a suitable parameter for the investigation, as outlined in Section 3. Ultimately it was concluded that, as the current implementation used the behaviours to steer the flies rather than redirect them, it was not necessary to add constraints on how they turn.

The weightings for separation and cohesion behaviour are built into the swarming behaviour code (Appendix C). The distance at which the separation behaviour is activated is also in this code, and is calculated as the current velocity of the fly divided by 10, which is a constant value decided on through preliminary experimentation.

In order to apply the changes to the variables as painlessly as possible, they are exposed to the user. "Sane" limits are enforced and the variables can only be set to discrete values within the ranges defined in Section 5.1. The source code for this phase of implementation is on the accompanying CD in the "Implementation Phase 1" folder, along with the executable used for testing and a read-me.

4.2 Implementing Effective Dynamic Obstacle Avoidance Behaviour

With the results from the previous round of testing having been used to select an ideal set of parameters for the swarming behaviour, these parameters are now hardcoded into the swarming behaviour component that each fly has. For this stage of implementation Dynamic Obstacles (which appear as cubes) have been added, which initially have set sizes and speeds that can be altered by the user in a similar fashion to the parameters in the previous stage of testing. The number of flies and maximum number of dynamic obstacles on-screen are variables that can also be altered by the user.

A "Scenery Manager" has been added that handles dynamic and static obstacles (although static obstacles aren't being investigated or used). This facilitates the dynamic addition and removal of the dynamic obstacles, as well as managing them and making them accessible. A new steering behaviour component has been added which makes flies aware of an obstacle at a distance and attempts to move them away from it. This takes the size of the obstacle into account, and has a variable weighting to help it take precedence over the other steering behaviours that may be present.

The ultimate goal of this stage of the implementation is to use the perfected swarming behaviour as a base for the additional dynamic obstacle avoidance behaviour. As a result all testing will be focussed on proving which parameters make for effective behaviour, and then proving this effectiveness when compared to other parameter values. When this behaviour is perfected for a range of dynamic obstacle sizes and speeds it will be considered complete, which will allow for the start of the next stage of implementation.

As with the previous stage, parameter value setting will be kept within “sane” limits and the actual values set will be discrete and within the range of these limits (all of these values and limits are outlined in Section 5.2). Some preliminary, unrecorded testing was done in order to establish these limits, but all further testing is used to gain results for the one metric being measured here – number of collisions between flies and dynamic obstacles. The code that records the number of collisions knows the difference between when a collision first occurs and when the same collision is still occurring, as no behaviour is present that keeps flies from flying around within the dynamic obstacles once collided with.

The code base is largely similar to that made in Section 4.1, and can be viewed on the accompanying CD in the “Implementation Phase 2” folder, along with the executable used for testing and a read-me. Appendix D has the current UML diagram for this phase of implementation (as *Fig D-1*), and the code for the dynamic obstacle avoidance behaviour is in Appendix E.

4.3 Creating Efficient Implementations

With now-perfected behaviour for both swarming and dynamic obstacle avoidance (in terms of effectiveness) the secondary objective of efficiency can be addressed. Using the best results from the previous two stages of implementation, a hard-coded, final version of the base swarming behaviour and dynamic obstacle avoidance behaviour have been coded. These behaviours are to be used as a base (and benchmark) for other implementations which have the efficiency boosting methods suggested in Section 2.6 applied to them.

The first implementation, labelled *Implementation I-1* is the same program as used in Section 4.2, but with the parameter and metric behaviour altered to measure FPS and the average behaviour calculation time per actor per frame. The only other alterations were an addition to the scenery manager, which now tidies up inactive dynamic obstacles at the end of every frame and the reorganising of the update loop to isolate the movement behaviour in order to measure it properly. Initially the behaviour calculation time was to be measured in every steering behaviour update run, but the times spent calculating are tiny fractions of a millisecond, which is the smallest value the timer class used can show. Consequently, the number of calculations per frame is instead used to divide the time spent calculating all of them in order to get a result. The UML diagram for this implementation is in Appendix F as *Fig F-1*. The code for this implementation is on the accompanying CD in the folder “Implementation Phase 3/Implementation I-1”, along with the executable used for testing and a read-me.

With this efficiency-testing-ready behaviour in place, some base values have been taken over a range of discrete values for number of flies and number of dynamic obstacles, to provide the aforementioned benchmark for further testing. Results were taken with object rendering on and off, so that there are "pure" results and applied results for this testing session. The only other 'parameter' is the efficiency boosting method used. For the purposes of efficiency testing, two alternative implementations have been investigated and created.

The first of these (*Implementation I-2*) aimed to improve the code for the behaviours as much as possible by streamlining. All the mathematical vector usage was altered to use length or distance squared where possible and lines of code were removed or compacted to reduce unnecessary computational complexity. The key change was to first centralise the movement and steering calculations to one loop, and then reduce the swarming steering behaviour down to an $(n^2 – n)/2$ problem by altering the loop to only check any flies that haven't previously been checked, whilst setting more results per iteration. The UML diagram for this implementation is in Appendix F as *Fig F-2*. The code for this new update loop is in Appendix G, but the original code, and the rest of the source code for this implementation is available on the accompanying CD in the folder "Implementation Phase 3/Implementation I-2", along with the executable used for testing and a read-me.

The second alternative implementation (*Implementation I-3*) swaps the current swarming behaviours "check-against-everything" method of operations for a blackboard-based nearest neighbour approach. This blackboard is a stored component for each object that keeps track of any objects within a radius of approximately 15 units (for efficiency purposes this implementation also uses square values for length and the like when possible, which reduces accuracy). This has the positive effect of reducing the amount of calculations done in the steering behaviour considerably. However, keeping the blackboards up-to-date has its own overhead, which is mitigated by being programmed efficiently (in the style of the update movement loop for Implementation I-2) and being only updated once every second. The code for the blackboard utilising swarming steering behaviour component and the blackboard manager update loop are in Appendix G. This has had no visual impact on the aesthetics of the behaviour. The UML diagram for this implementation is in Appendix F as *Fig F-3*. Likewise, the source code for this implementation is available on the accompanying CD in the folder "Implementation Phase 3/Implementation I-3", along with the executable used for testing and a read-me.

All three implementations have been measured within the same range of variables under identical circumstances (same machine, etc.) and the same two metrics (Average FPS and average time taken for behaviour calculation per actor per frame) have been taken.

5 Results and Analysis

The results collected from the previous section are examined, compared and analysed here. Each sub-section begins by listing the metrics and parameters used as well as setting the ranges and increments of the parameters. The results are then pointed to in the relevant appendix and analysed, including comparisons to expected trends and outcomes for the results.

5.1 Effectiveness Results for Swarming Behaviour

The metrics being measured in this section are:

- Average speed of the swarm
- Average distance from the centre of the swarm
- Average speed within the average distance
- Average speed outside of the average distance

These four metrics provide a profile for the movement of the swarm on average, which should be sufficient for comparing effectiveness when combined with a visual appraisal. As the visual appraisal is entirely subjective, a formalised scale is useful in quantifying this opinion. A rating out of 10 has therefore been applied to the visual aspect of the demonstration, with 0 being "looked nothing like swarming behaviour" and 10 being "this is what is expected of swarming behaviour".

The parameters being altered in order to tweak the behaviour are as follows:

- Number of flies in swarm
  - Multiples of 40 ranging from 40 to 400
- Minimum speed of the flies within the swarm
  - Whole numbers between 10 and 19 (inclusive)
- Maximum speed of the flies within the swarm
  - Whole numbers between 11 and 20 (inclusive)
  - Always at least 1 greater than minimum speed
- Weighting of cohesion behaviour
  - Multiples of 0.1 between 0.1 and 3.0 (inclusive)
- Weighting of separation behaviour
  - Same range and increments as cohesion weighting.

Results in this section were collected by giving each parameter value combination 5 seconds to stabilise and then taking a pair of screenshots. The screenshots were used to collect the values for the table and analyse the behaviour to get a visual appraisal, as all of the metric values update every second to reflect the state of the behaviour more accurately at that time. Whilst this does provide a high risk of anomalous values, overall trends in the values will be sufficient to mitigate any anomalies that do arise. The gathered results for this section are in Appendix H as tables labelled *Table H-#.*

Expected results for *Table H-1* were that larger speed ranges would result in more varied, interesting and aesthetically appealing swarming behaviour, with a maximum speed of 20 (the highest being tested) being ideal, and a minimum speed near 10 best complementing this. However, the values which produced the best swarming behaviour (*Table 5.1 – 1*) actually suggest

(barring anomalies) that the best candidate for a maximum speed is either 17 or 18, and that the best minimum speed would be found in the range of 11-14. Further analysis of the metric results suggest that the best average speed is between 14 and 15.5 and that a good swarm of 40 flies would have flies maintaining an average distance of around 7-9 units from the centre.

Minimum Speed	Maximum Speed	Average Speed	Average Distance to Centre	Avg. Speed within Avg. Distance	Avg. Speed out of Avg. Distance
11	17	13.5	8.2	14.1	12.9
11	18	14.3	8.4	14.3	14.3
12	15	13.9	6.5	14.0	13.7
12	17	14.8	7.2	14.8	14.8
12	18	14.9	9.4	15.0	14.7
13	15	14.1	8.8	14.0	14.2
13	17	15.0	8.5	15.9	13.8
14	16	15.1	7.8	15.0	15.2
14	17	15.6	8.1	15.6	15.6
14	18	15.5	9.2	15.5	15.6

Table 5.1 – 1: All of the values from Table H-1 *which gained a subjective appearance rating of 8 (highest given in these results).*

Table H-2 was expected to prove that higher cohesion values (around 2.0) and fractional separation weightings will work best together (above 0.5, below 1.0) to produce better behaviour. The data in *Table 5.1 – 2* would suggest that the difference between the weightings need not be so extreme at the speeds tested at, however. If the results for the Separation Weighting of 1.9 are discarded as anomalous, then separation works best in the range of 1.2-1.6, with Cohesion being weighted at 1.9-1.5. Other combinations of these values within those ranges should be tested with the speed ranges in *Table 5.1 – 1* to best discern a strong combination. The average distance to centre was notably lower this time around, largely because of the stronger cohesion weightings.

Separation Weighting	Cohesion Weighting	Average Speed	Average Distance to Centre	Avg. Speed within Avg. Distance	Avg. Speed out of Avg. Distance
1.2	1.9	14.9	6.5	14.7	15.0
1.3	1.8	14.4	6.0	14.6	14.3
1.4	1.7	14.9	6.6	15.1	14.8
1.5	1.6	15.5	6.4	15.3	15.8
1.6	1.5	15.9	6.3	16.1	15.4
1.9	1.2	14.6	6.1	14.5	14.6

Table 5.1 – 2: All of the values from Table H-2 *which gained a subjective appearance rating of 8 (highest given in these results)*

Trends detected from *Table H-1* and *Table H-2* (Appendix H) show that high speeds or high separation weightings combined with lower cohesion values very quickly remove any semblance of swarming from the behaviour.

However very tightly clustered swarms due to very low speeds, low separation weightings and/or high cohesion weightings are not as visually unappealing. This implies that cohesion and the average distance to the centre of the swarm are the most aesthetically important aspects. With 40 flies, the distance should stay in the range of 6-9 a vast majority of the time for best results and an average speed of 14-16 complements this.

Using this information, 9 parameter value combinations and metric results were produced as *Table H-3*. The results show that a maximum speed of 18 works best and a minimum speed of 14 is most successful in regulating the steering behaviour. Likewise a cohesion value of 1.7 and a separation weighting of 1.4 were generally involved in the best results, which ultimately means that an average speed of 15-16 works best when combined with an average distance of 7 units from the centre for a swarm of 40 flies can be considered the best balance. *Table H-4* shows how the resulting metrics from the parameter values scale over a range of swarm populations. The behaviour scales well until 200 flies, where afterwards the density of the swarm detracts from the appearance, with the average distance from the centre being reduced somewhat by the slightly higher cohesion weighting. However, for the purposes of this investigation the swarming behaviour can be deemed complete, as the avoidance of dynamic obstacles is the real focus.

5.2 Effectiveness Results for Dynamic Obstacle Avoidance Behaviour

Taking all parameters for swarming behaviour (aside from number of flies) as fixed values, this section instead focuses on altering parameters for the flies to avoid dynamic obstacles. The parameters open to modification are:

- Number of flies in the swarm
  - Multiples of 30, ranging from 30-300
- Distance at which flies become aware of dynamic obstacles
  - Multiples of 4 between from 4-32 (inclusive)
- Weighting of dynamic obstacle avoidance behaviour
  - Multiples of 0.1 between from 0.1-4.0 (inclusive)
- (Maximum) Number of dynamic obstacles on-screen
  - 1-5 (inclusive)
- Size of dynamic obstacles
  - Multiples of 2 between 2-20 (inclusive)
- Speed of dynamic obstacles
  - Multiples of 3 between 3-30 (inclusive)

The metric being measured is number of collisions between the flies and the dynamic obstacles, although a subjective visual appraisal will also be taken in order to give the results some context - flies all successfully avoiding an obstacle when they really shouldn't be capable of it would be worse than some being hit, after all. Appendix I has the results tables for this section.

Table I-1 lists the results for the full range of weightings for the steering behaviour against the full range of distances the flies will become aware of a dynamic obstacle. A higher-end middle ground was expected to produce the

best results, with the weighting being in the 2.5-3.5 range, and a distance value of 16 being the expected preferred values for minimal collisions. However, *Table 5.2-1* suggests that the weighting range of 2.1-2.5 was actually better, and the distance value produced satisfying results from around 8-16 units.

Avoidance Behaviour Weighting	Distance of Object-Awareness	Number of Collisions	Appearance Rating (out of 10)
2.1	4	20	5
	8	7	8
	12	3	10
	16	3	9
	20	1	7
	24	1	5
2.2	4	19	5
	8	6	9
	12	4	10
	16	1	10
	20	0	6
2.3	4	9	5
	8	4	9
	12	0	10
	16	2	10
	20	1	6
2.4	4	17	5
	8	4	9
	12	0	10
	16	0	10
	20	3	6
2.5	4	8	5
	8	3	9
	12	0	10
	16	0	10
	20	0	6

Table 5.2-1: The values from Table I-1 *that form the area of interest, showing where the best results are for combinations of avoidance behaviour weighting and object-dodging range.*

A distance value below 8 was too small a reaction time, and gave the impression that the obstacles were almost invisible, or of little interest. Distance values above 16 produced highly exaggerated reactions, which would either then be overridden by the swarming behaviour when combined with a low rating, or would produce huge looping arcs of movement as the flies gave the obstacles an unreasonably wide berth.

The weighting values below 2.1 tended to result in too many collisions to make the flies dodging behaviour appear convincing, and values above 2.5 very quickly made the flies appear to be reacting too quickly, with the higher

extreme meaning that even with a reaction distance of 4 units the flies would almost universally avoid collision. The range highlighted in *Table 5.2-1* produced behaviour that appeared acceptable at ranges of 12-16 units although *Table I-1* would suggest that a lower value works best, so 12 will be selected as the distance value. The appearance ratings for a weighting between 2.2 and 2.5 are all very similar, although only 2.4 and 2.5 produced no collisions at a distance of 12 and above (therefore implying reliability). As the middle value of 2.45 requires rounding up to 2.5, 2.5 shall be the value chosen as the weighting to be tested further.

Table I-2 shows the selected parameter combination being compared against a varying number of obstacles, which was expected to make little difference to the number of collisions, as the behaviour has been balanced for the size and speed of the obstacles present. However, the results imply that the selected weighting isn't quite high enough, as the number of collisions generally rose with the number of boxes. As a result of this, the selected weighting will be boosted to 2.7 and retested.

Table I-3 is a repeat of the same test as for the results in *Table I-2*. The expected outcome was that the marginally increased weighting will produce more stable results in more complicated environments. This prediction proved correct, with the behaviour performing much better, but not proving infallible, with larger numbers of dynamic obstacles.

Table I-4 shows the results for 60 flies against two objects of a variable size. The expected outcome here is that the larger shapes are harder to avoid and should result in a few collisions, whilst smaller shapes should be very unlikely to score any hits. The general trend is that shapes with a radius of 10 or below proved easy to avoid (barring anomalies), whilst any radius from 12-20 proved more consistently difficult to avoid, particularly with radii above 16. The results here largely conform to the prediction and the appearance is generally very acceptable.

Table I-5 keeps the size of the two shapes static (at 10) and instead tests the range of speed for the obstacles. Flies are expected to be able to dodge shapes relatively easily up until the shape reaches their own maximum velocity (of 18), where collisions should become more frequent. The results match this prediction well, with both extremes being too extreme for the flies (3 is too slow to pose any threat, and 30 is so fast that some flies were hit by both boxes).

The previous five results tables and analysis show that the behaviour works well in the presented environment with a weighting of 2.7 and a reaction distance of 12 when the obstacles are not too large or moving too fast. The behaviour scales well for the number of obstacles and the appearance is very satisfying overall. Consequently these values can be set as standard and the implementation advanced in order to examine efficiency. A further conclusion to be drawn here is that steering behaviour for dodging dynamic obstacles **effectively** is indeed a suitable addition to a Boids Model based complex movement behaviour, such as swarming.

5.3 Efficiency Results

With the parameter values for the behaviours now set at fixed values, the three implementations discussed in Section 4.3 have been subjected to efficiency testing. The two metrics being taken are Average Frames per Second (FPS) and Average Time Taken to Calculate Behaviour per Frame per Object (in milliseconds), which represent the overall program efficiency and behaviour efficiency respectively.

Efficiency metric values have been taken with and without rendering for each implementation, whilst the actual testing parameters are the number of flies and number of dynamic obstacles. Dynamic obstacles were randomised for size (range 6-16) and speed (range 9-24), as these parameters shouldn't have any effect on performance from an efficiency stand-point, and these are the middling ranges from the testing in Section 5.2 and proved to be the most reasonable. The rendering isn't programmed to be efficient in any of the implementations, so the "applied" values are limited more by that than the number of flies and obstacles present. Therefore, the range for number of flies with rendering was multiples of 50 between 50 and 250 (inclusive), whilst the range for number of flies without rendering was the multiples of 100 between 100 and 1000 (inclusive). The number of dynamic obstacles was within the range 0-5 and was tested against all values for number of flies.

All values for FPS were expected to be below 60, as the machine the testing was done on is capped at 60 and usually shows an average FPS of approximately 59.5 at best. The machine is a gaming laptop, which has modest specifications for 2011, having been assembled in 2007:

- System:
  - Microsoft Windows XP
  - Professional
  - Version 2002
  - Service Pack 2
- Computer:
  - Intel® Core™2 CPU
  - T5500 @ 1.66GHz
  - 1.66 GHz, 2.00 GB of RAM
  - NVIDIA GeForce 8400M G

Appendix J has the tables of results for this section of testing. *Table J-1* features the results for all 3 implementations with rendering. The expected results here are for the standard implementation (I-1) to have the worst results at higher numbers of flies and obstacles for both metrics. Implementation I-2 should have largely improved values for both FPS and calculation times at the higher values due to the improved programming and the reduction of the impact of the number of flies. I-3 (the blackboard-based implementation) is expected to have little-to-no improvement for the FPS as the calculations saved in the behaviour will be used for keeping the blackboards up-to-date. It is expected to handle the behaviour calculations faster for the same reason, however.

Implementation I-1 proved efficient with rendering enabled, with all FPS statistics being approximately 59, and therefore remaining roughly at the maximum for the testing machine. The time spent calculating the behaviour was considerably more indicative with the (very small) times slowly increasing over the number of flies. Oddly (or perhaps anomalously) the number of dynamic obstacles being tested against didn't produce any clear upwards correlation with calculation times.

Implementation I-2 produced similar FPS figures to I-1. However the figures for behaviour calculation time demonstrate the improvement in efficiency. Typically, the results are around 50-75% smaller than those for I-1. Unfortunately the results for 50-100 flies for I-2 can all be discounted as anomalous – seemingly the program can't handle such low average calculation times and display them in any consistent or meaningful fashion. At 200-250 flies the time difference is clear and reliable however, demonstrating some of the potency of the efficiency-minded approach.

Implementation I-3 produced consistently marginally lower FPS results when compared to the previous two implementations, which was unexpected, but easily explained due to the additional overhead of keeping the blackboards up-to-date. The behaviour calculation times show an improvement over I-1, whilst being weaker than I-2s results. One main observation from the results is that the calculation times fluctuate considerably, but stay relatively low throughout. The implication here is that this method scales well over larger numbers.

Table J-2, which is for without rendering, demonstrates a much larger range of testing values, and should produce more definitive results for FPS based purely on calculation speeds. Ultimately the results are expected to mirror those from *Table J-1*, but with a more gradual decrease in performance for each implementation.

As expected, *Table J-2* highlights the problem with Implementation I-1, which is that it is an n^2 problem as far as big 'O' notation is concerned. Results from 100-300 flies for FPS are largely similar, but as the average behaviour calculation time increases linearly for each hundred flies a threshold is approached at 400 flies, which dips the frame rate by a frame or two. After this the frame rate plummets between 500 and 700 flies, where it then proceeds to bottom out due to the increased duration of frames meaning the FPS slowly approaches 0. Results for 1000 flies are steady at approximately 11.4 frames per second. The calculation for dynamic obstacle avoidance is shown to have minimal impact on the frame rate even with 1000 flies being calculated, as the calculation is once per fly, per obstacle. Consequently it can be concluded that the steering behaviour for swarming is the most in need of streamlining.

For Implementation I-2 this table really shows the improvements made, as the FPS only starts to drop below the standard ~59 at 1000 flies, as opposed to the 400 flies in I-1. The reason for this is shown clearly in the figures for behaviour calculation time. Examining the table in depth, it can be seen that,

on average, the improvement is somewhere in the range of 70-90% time saved. Therefore in terms of pure efficiency, Implementation I-2 is a huge improvement over I-1. However, a mitigating factor in this is that two of the scaling factors for determining distances at which behaviours become active did need to be tweaked after converting the behaviour in order to make it visually similar to the original implementation. Likewise the use of squared values does reduce accuracy considerably when working out distances and scaling factors. The biggest detractor from this promising result is simply that the implementation style does not lend itself well to use in games however, as normally games would tend to uses multiple different types of character with different kinds of behaviour. Outside of an experiment such as this, the method would likely cause more issues than it would solve by being so efficient. The Boids model works as a localised method because it is so easily portable, so taking this away in the interests of efficiency is something of a double-edged sword.

This table confirms the assumptions made from the results for Implementation I-3 in *Table J-1*, as the FPS value at 1000 flies are comparable to those for Implementation I-2, but with considerably less stable results over time. Depending on the positioning of the flies, I-3 can have the edge over I-2 at fly populations starting at 700. The value for average calculation times in particular demonstrates what appear to be random fluctuations. These fluctuations represent times when the flies have clustered close together or have been shepherded into a tight space by the dynamic obstacles, resulting in a much higher number of checks-per-fly than if they are left in a stable swarm, undisturbed. A major benefit with this version of the implementation is that the methodology used to streamline the number of calculations does fit well into a game system, and the update timing and radius of the flies' local area can be altered.

Comparing the values in *Table J-2* for the implementations I-2 and I-3 using the *Graphs K-3* and *K-4* in Appendix K the two implementations would appear approximately evenly matched for efficiency at the higher extreme values. However, at lower values (both with rendering active and inactive) the centralised approach to handling movement and steering behaviour is clearly more efficient. Whilst it has been pointed out that I-3 is an appropriate method for a game scenario, and I-2 is best left to specialist implementations such as the artefacts for this report, the lack of stable results from I-3 should be taken into consideration. In theory a tight swarm, or one that uses large radii for blackboard population, could easily be just as inefficient as Implementation I-1. Given that both versions also sacrifice accuracy in distance measurement for efficiency, it would be fair to say that neither method is truly ideal.

This stage of the investigation has, however, shown a clear benefit in using efficiency-boosting methods for swarming (and therefore flocking and other steering-based) behaviours. As stable, reproducible results are generally more reliable in a game scenario, Implementation I-2 can be considered the most useful efficiency-boosting method investigated here, but further investigation into other approaches to the problem is encouraged.

6 Conclusion

This investigation is ultimately to prove that games AI can be aware of dynamic obstacles and can indeed effectively avoid them in a convincing, fallible fashion. Even in large, relatively complicated groupings a simple additional behaviour for the AI can be used to achieve the desired effect.

The implementation from Section 4.1 provided this relatively complicated grouping, in the form of an effective swarming behaviour. The effectiveness is important in order to ensure that the behaviour could be applied to, and accepted as part of, a game. These swarming flies conform to the criteria laid out in Section 2.1.1 and had their behaviour perfected in Section 5.1, using the results laid out in Appendix H. This steering behaviour-based swarm, which works using separation and cohesion behaviour similar to that seen in the Boids model (Reynolds, 1999), built from an adaptation of the code in "Swarm" (Turner, 2010) certainly could serve the purpose of a focussed swarm of enemies in a First Person Shooter game, for example.

With a behaviour established, the simulated flies have been augmented in Section 4.2 to have dynamic obstacle avoidance behaviour. This behaviour is an additional steering behaviour, which was made as effective as possible in Section 5.2 using the results from Appendix I, whilst kept from becoming infallible by an appearance rating being used as a check against the metrics. This behaviour working within acceptable parameters, even when combined with a swarming model which is passably accurate for flies can be used as evidence to draw the conclusion that games AI can indeed be fully aware of dynamic obstacles and acceptably and effectively avoid them.

However, this conclusion is worthless if the behaviours can't be applied to games properly. Consequently Section 4.3 provided two alternative implementations, both using the behaviour parameters deemed best from the previous results. Testing in Section 5.3 shows that only in a swarm of well over 100 objects would the behaviour have a real, visible impact on game performance, and that measures such as using blackboards to provide nearest-neighbour information and code centralisation can improve the number of objects being simulated at a decent frame rate enough that most games would never use enough to cause issues, assuming rendering was handled efficiently. The best method for improving the efficiency of the behaviour was expected to be the efficiency-minded, centralised approach to movement and steering behaviour calculation. The actual conclusion here is that it was merely the best out of the two methods investigated, and only by a marginal amount. Further investigation into efficiency boosting methods is recommended, as neither of the examined methods can be considered ideal.

The final conclusion is that the methods used during this investigation would indeed be suitable for producing steering behaviour-based games AI capable of avoiding dynamic obstacles, and could be adapted for modern 3D games using obstacles of variable shape and size.

7 Evaluation

This report has successfully proven the point it set out to investigate, having covered a wide range of methodology, considerations and implementation issues. Also in its favour, the investigation was kept within a reasonable scope, by limiting the behaviour being adapted in the research sections through analysis and elimination of a number of options. Keeping the implementation in two dimensions (although rendered in 3D), and treating the flies as points in space also served to streamline the investigation. Finally, the discovery of Turner's Java swarming behaviour was quite fortunate, as it provided a strong starting point for the implementation phases of this project.

Conversely, the key weakness of this project is in the lack of hard data for the base behaviour being emulated (that of a swarm of flies). Whilst metrics and subjective opinion were created to fit with a measurable definition which had a selection of criteria, a lot of the measurements used for the results in Appendix D are ultimately open to interpretation. Fortunately the focus of the investigation is the dynamic obstacle avoidance behaviour, which was easier to measure on an effectiveness scale and produced results that are a lot easier to interpret. The artefacts from Implementation Phase 3 (Section 4.3) also displayed a weakness, which was present in previous phases, but never exhibited. This weakness was that altering the limits of a parameter when previously at a limit would result in the program still assuming a limit was being hit until the parameter value changed. This merely slowed testing a little however.

Possible improvements for the project include:

- Working with a more efficient rendering system and/or using better rendering methods in order to gain more meaningful results with rendering active.
- Using a different behaviour as base – one that can be easily accepted and defined, with hard evidence to back it up.

Suggestions for further work:

- The creation of a full 3D version of this demonstration. This has been tested briefly without documentation, and the swarming behaviour does indeed adapt very well to the extra dimension. However a suitable framework would still need to be established around it to truly make use of it.
- A more in-depth research into efficiency, using a wider range of metrics and methods.
- The combination of the steering behaviour with a purpose-driven games AI that incorporates higher level objective algorithms such as the A* algorithm. Objects would attempt to follow a static path and then steer around any dynamic obstacles they come across, whilst avoiding any static hazards or slowing their journey too much.

References and Bibliography

Alexander, B. (2006) “3.1 Flow Fields for Movement and Obstacle Avoidance”. In: Rabin, S, ed 2006. *AI Game Programming Wisdom 3*. Boston, MA: Charles River Media

Amor, H. et al. (2006) “3.5 Fast, Neat, and Under Control: Arbitrating Between Steering Behaviours”. In: Rabin, S, ed 2006. *AI Game Programming Wisdom 3*. Boston, MA: Charles River Media

Animal Planet (2009) “Weird, True and Freaky: Fly Swarm”. http://animal.discovery.com/videos/weird-true-freaky-fly-swarm.html [01/04/11]

Brownlee, J. (2002) “Finite State Machines (FSM)” http://ai-depot.com/FiniteStateMachines/ [02/04/11]

Buckland, M. (2005) *Programming Game AI by Example*. Plano: Wordware Publishing, Inc.

Cresswell, J. (2010) “OpenGL First Person Shooter” http://www.phobus.servegame.com/assignment2.html [04/04/11]

Cruz, A. (2008) “Fuzzy State Machines” http://equipe.nce.ufrj.br/adriano/fuzzy/transparencias/fsm/fsm.pdf [02/04/11]

Delorenzo, D. (2005) “2.7 Avoiding Dynamic Obstacles and Hazards” http://www.cse.lehigh.edu/~munoz/CSE497/classes/DeLorenzoObstacles2.7.ppt [02/04/11]

Fairclough, C. et al. (2001) “Research Directions for AI in Computer Games” http://www.scss.tcd.ie/publications/tech-reports/reports.01/TCD-CS-2001-29.pdf [02/04/11]

Flensbak, J. (January 2007) “Flock Behavior Based on Influence Maps” ftp://130.225.96.5/diku/image/publications/Flensbak060125.pdf [10/04/11]

Finkelstein, L. (1982) "Theory and Philosophy of Measurement". In: Sydenham, P, H ed 1982. *Theoretical Fundamentals, vol. 1, Handbook of Measurement Science*, Chichester: John Wiley & Sons

Hansson, N (2010), “Influence Maps I”. *Game School Gems* http://gameschoolgems.blogspot.com/2009/12/influence-maps-i.html [10/04/11]

Johnson, G. (2004) “2.7 Avoiding Dynamic Obstacles and Hazards”. In Rabin, S, ed 2004. *AI Game Programming Wisdom 2.* Hingham, MA: Charles River Media, Inc.

Kaner, C., Bond, W. B. (2004) “Software Engineering Metrics: What Do They Measure and How Do We Know?” http://www.kaner.com/pdfs/metrics2004.pdf [03/04/11]

Lager, P (2006) “Session 19”. *Programming For Games* http://data.bolton.ac.uk/staff/pkl1/UOB2/prog4games/index.php?n=ProgrammingForGames.Session19 [08/04/11]

Lester, P. (2005) “A* Pathfinding for Beginners”
http://www.policyalmanac.org/games/aStarTutorial.htm [08/01/11]

Miller, P. (2007) “The Genius of Swarms”
http://ngm.nationalgeographic.com/2007/07/swarms/miller-text [18/02/11]

Preuss, M et al. (December 2008) “Intelligent Group Movement and Selection in Real Time Strategy Games”. *Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods.*
https://eldorado.tu-dortmund.de/bitstream/2003/26162/1/25508.pdf [01/04/11]

Reynolds, C. W. (July 1987) “Flocks, Herds and Schools: A Distributed Behavioural Model”. *Computer Graphics*.
http://www8.cs.umu.se/kurser/TDBD12/HT02/papers/ReynoldsBoids1987.pdf [10/04/11]

Reynolds, C. W. (1999) “Steering Behaviours for Autonomous Characters”.
http://www.red3d.com/cwr/steer/gdc99/ [10/04/11]

Reynolds, C. W. (2000) “Boids (Flocks, Herds and Schools: A Distributed Behavioural Model”. http://www.red3d.com/cwr/boids/ [10/04/11]

Reynolds, C. W. (1997-2004) “Steering Behaviours for Autonomous Characters”.
http://www.red3d.com/cwr/steer/ [08/01/11]

Reynolds, C. W. et al. (2002-2006) “OpenSteer: Steering Behaviors for Autonomous Characters”.
http://sourceforge.net/project/showfiles.php?group_id=77546 [08/01/11]

The Free Dictionary (2011) “definition of swarm by The Free Online Dictionary, Thesaurus and Encyclopedia” http://www.thefreedictionary.com/swarm [01/04/11]

Thunderhead Engineering (2009) “Pathfinder Technical Reference”,
http://www.thunderheadeng.com/downloads/pathfinder/Pathfinder_tech_ref.pdf [02/04/11]

Turner, A. (2010) “Swarm”. *Open Processing*
http://www.openprocessing.org/visuals/?visualID=7499# [03/04/11]

Üngör, A (2001) “An Alignment Algorithm for Anisotropic Meshes with Non-Uniform Flow Fields”. http://www.cise.ufl.edu/~ungor/abstracts/numgrid00.html [10/04/11]

v. d. Sterren, W. (2002) “Tactical A* Explorer”.
http://www.cgf-ai.com/products.html#tacastarexplorer [08/01/11]

Appendix A

The following is a transcription of the e-mail conversation between Craig Reynolds and the author regarding OpenSteer:

“Hello!

I am a third year student at the University of Bolton (UK) studying Computer Games Software Development. The final year Project module for this course requires us to produce a suitably complex program which has academic merit. We are encouraged to work on our areas of interest for this, as it involves a lot of work and commitment throughout the year.

My chosen project is essentially to model a swarm (of flies, for example) using work derived from your Boids model, which will eventually exhibit path-finding, steering and dynamic obstacle avoiding behaviour. I'm currently in the research stage, and whilst reading through ftp://130.225.96.5/diku/image/publications/Flensbak060125.pdf which also worked with the Boids model, I came across a link to OpenSteer.*

Having checked it out, I think some of the source would be incredibly useful in the implementation phase of this project. I'm not certain if my project is large enough in scale to bother you with, but just in case I felt I should let you know that I'm hoping to use some of the code as an example when producing my own derivative model.

Thanks for your time and all of the work you've put into this area of research,

-James Cresswell

**Jonas Flensbak's thesis resulting from a Bachelor project in computer science at the Department of Computer Science, University of Copenhagen (DIKU), Denmark.*

Hi James. I read your message a couple of times trying to figure out what you were asking for. But you weren't, right? You were just telling me that you will be referencing my code as you work on your project. You are probably the kind of person who reads documentation and follows instructions, aren't you?! I realized you were probably motivated by the place in the doc where it says "However please contact us before planning to integrate OpenSteer into any large project." That was originally just to make sure that no commercial game developers mistook this still-unpolished library as bullet-proof production quality code.

I hope starting from OpenSteer gets your project going sooner. Ideally it will allow you to push out into areas that are not already supported by OpenSteer

Thanks for the link to the Flensbak paper, I don't remember seeing that before.

Best,
Craig

Thanks for your reply!

You summed it up quite well there, I was following the documentation, as I was uncertain as to what a large project is, relative to OpenSteer. Sorry for any confusion, and thanks for the support.

-James Cresswell”

Appendix B

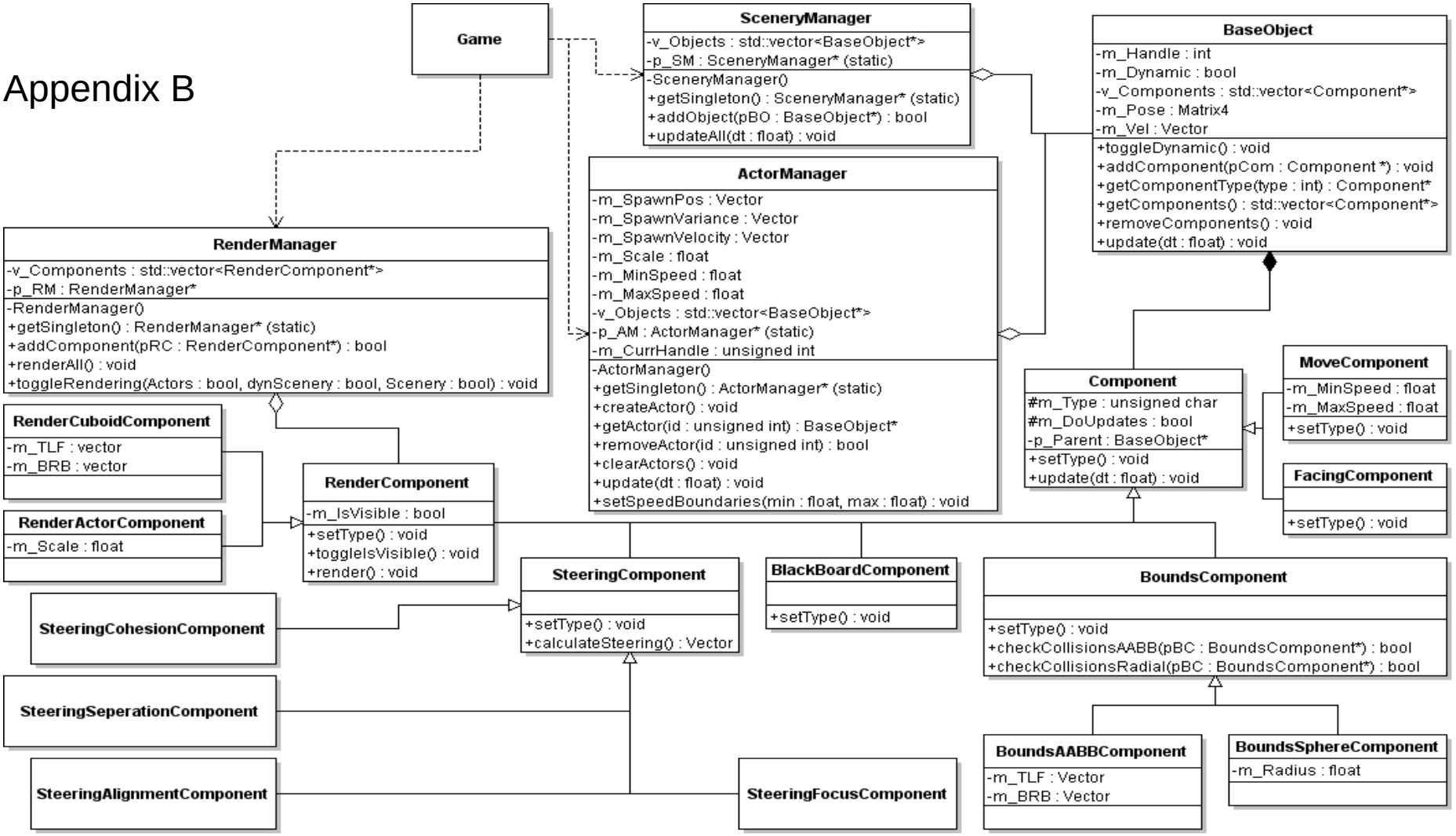


Fig B-1: A simple UML diagram showing the original design for Section 4.1.

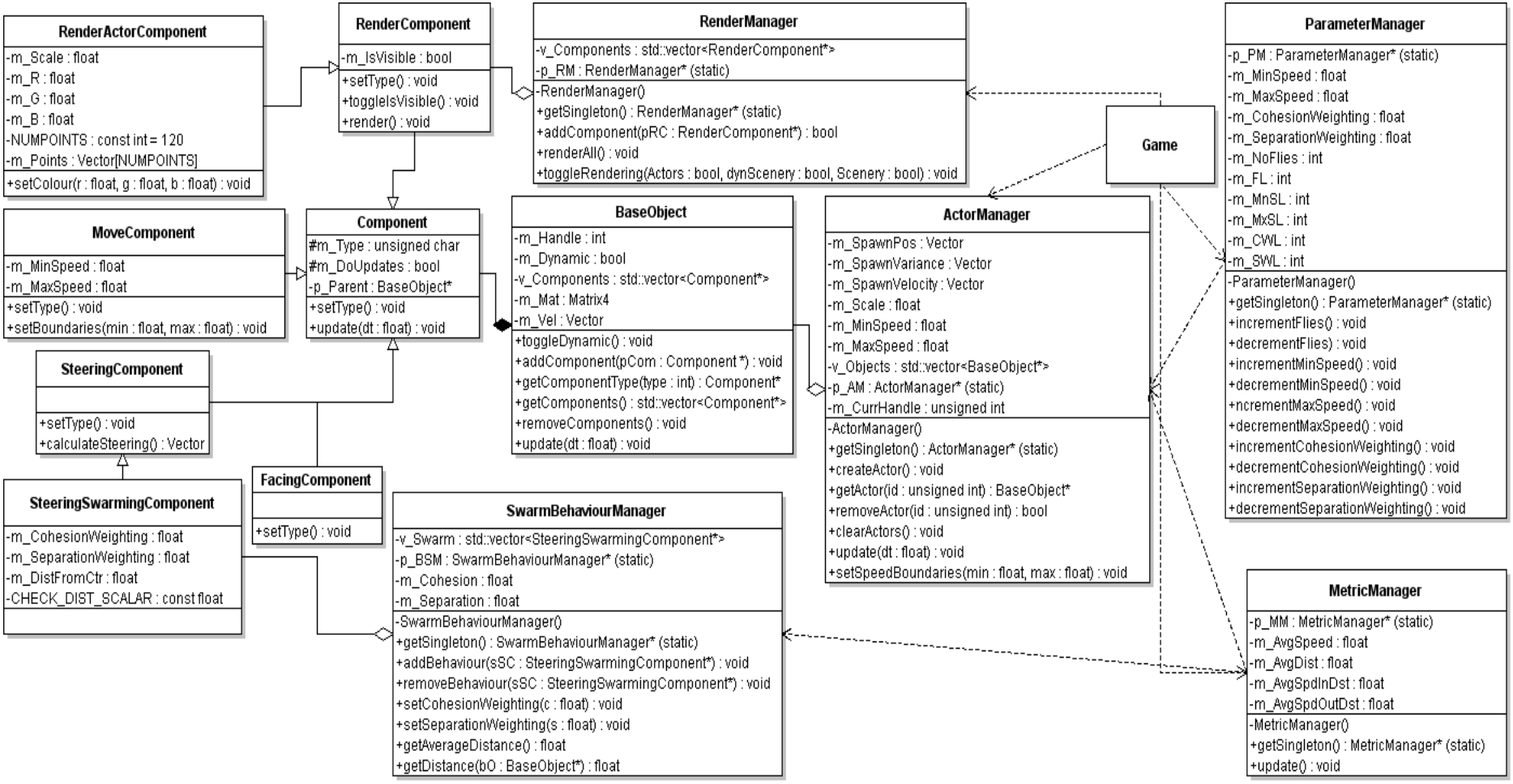


Fig B-2: A UML diagram showing the final, used classes for the implementation in Section 4.1.

Appendix C

Code listing for the calculateSteering() function in the SteeringSwarmingComponent class, as it appears in Section 4.1:

```
Vector SteeringSwarmingComponent::calculateSteering(){
	Vector suggestion(0.0f);
	//calculate central focus of the swarm
	Vector centre(0.0f, 0.0f, 0.0f);
	unsigned int i;

	for(i = 0; i < ActorManager::getSingleton()-
>getCurrentHandle(); i++){
		if(!ActorManager::getSingleton()->getActor(i)) continue;
		centre += ActorManager::getSingleton()->getActor(i)-
>getPosition();
	}

	centre *= (1.0f / i);

	Vector mouse = Vector(((float)g_Game.m_mouse.x / 3), -
((float)g_Game.m_mouse.y / 3));
	centre = mouse - centre;

	Vector centredir = centre - getParent()->getPosition();
	m_DistFromCtr = centredir.Length();

	//work out which fly is closest, and what the distance is
	int closest = -1;
	float closestdist = 1000.0f; //arbitrary value

	for(i = 0; i < ActorManager::getSingleton()-
>getCurrentHandle(); i++){
		if(!ActorManager::getSingleton()->getActor(i)) continue;
		Vector v = ActorManager::getSingleton()->getActor(i)-
>getPosition() - getParent()->getPosition();
		float d = v.Length();

		if(ActorManager::getSingleton()->getActor(i) !=
getParent() && d < closestdist){
			closest = i;
			closestdist = d;
		}
	}

	//fly to centre if not too close to the nearest fly
	if(closestdist > getParent()->getVelocity().Length() /
CHECK_DIST_SCALAR){
		centredir.Normalize();
		suggestion += centredir * m_CohesionWeighting;
	}
	else{ //dodge flies
		Vector closestdir = ActorManager::getSingleton()-
>getActor(closest)->getPosition() - getParent()->getPosition();
		closestdir.Normalize();
		suggestion -= closestdir * m_SeparationWeighting;
	}

	return suggestion;
}
```

Appendix D

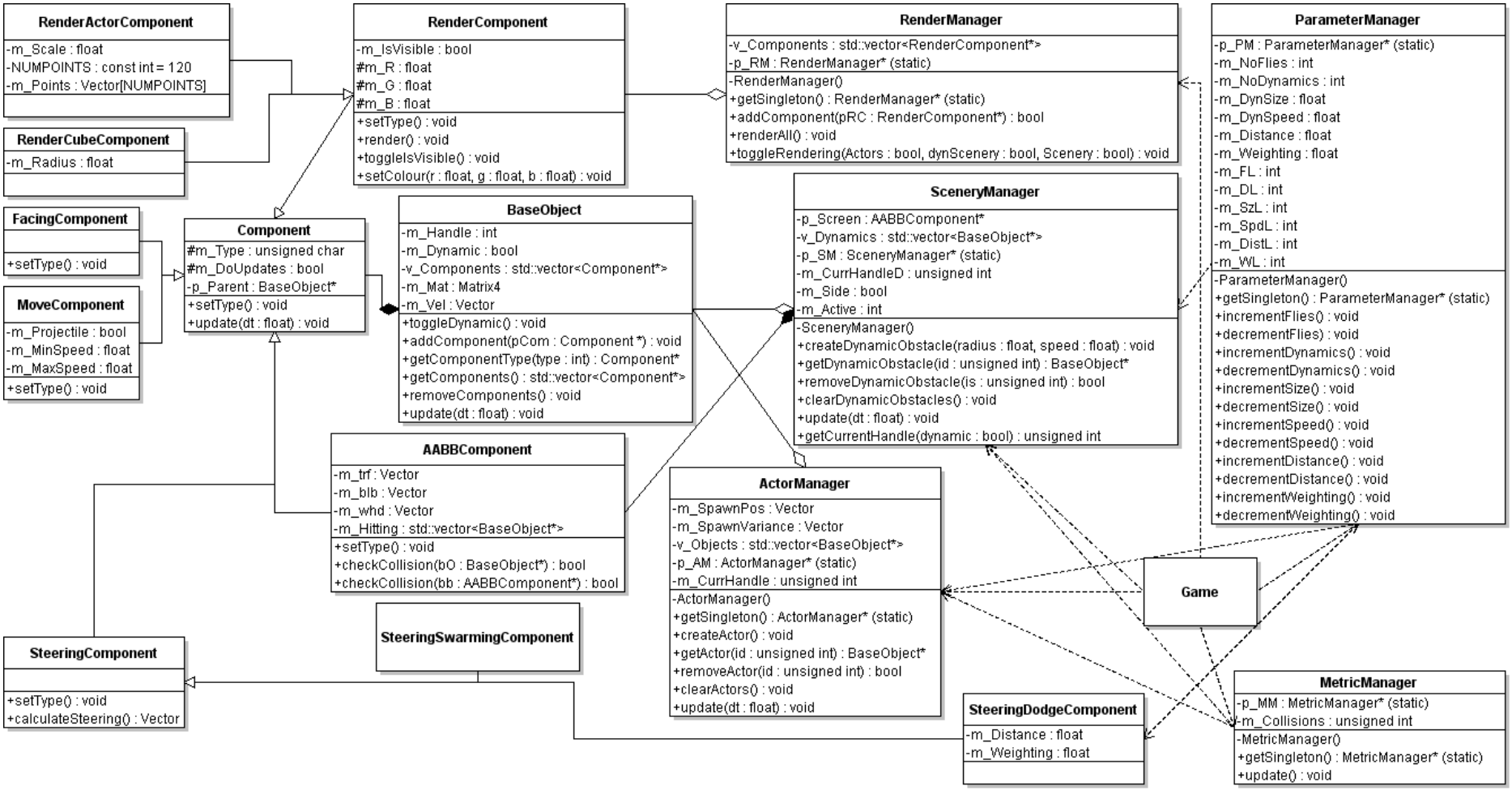


Fig D-1: Final UML Diagram for Implementation Phase 2, as implemented in Section 4.2

Appendix E

Code listing for the calculateSteering() function in the SteeringDodgeBehaviour class as it appears in Section 4.2:

```
Vector SteeringDodgeComponent::calculateSteering(){
	Vector suggestion(0.0f);

	for(unsigned int i = 0; i < SceneryManager::getSingleton()->getCurrentHandle(); i++){
		BaseObject* bO = SceneryManager::getSingleton()->getDynamicObstacle(i);
		if(!bO || !bO->getIsDynamic()) continue;

		AABBComponent* bb = (AABBComponent*)bO->getComponentType(C_TYPE_BOUNDS);
		if(!bb) continue;

		Vector distance = bO->getPosition() - getParent()->getPosition();

//As we're only dodging in two axes, and the obstacles are all cubes,
//we subtract the diagonal on the x/y plane to the distance we're
//calculating - which means we take magnitude into account when
//saying how far we are from the obstacle.
		if(distance.Length() - sqrt((bb->getWHD().x * bb->getWHD().x) + (bb->getWHD().y * bb->getWHD().y)) > m_Distance) continue;

		if(distance.x > 0) distance.x += bb->getWHD().x;
		else distance.x -= bb->getWHD().x;
		if(distance.y > 0) distance.y += bb->getWHD().y;
		else distance.y -= bb->getWHD().y;

		suggestion -= distance;
	}

	if(suggestion == Vector()) return Vector();

	suggestion.Normalize();
	suggestion *= m_Weighting;

	return suggestion;
}
```

Appendix F

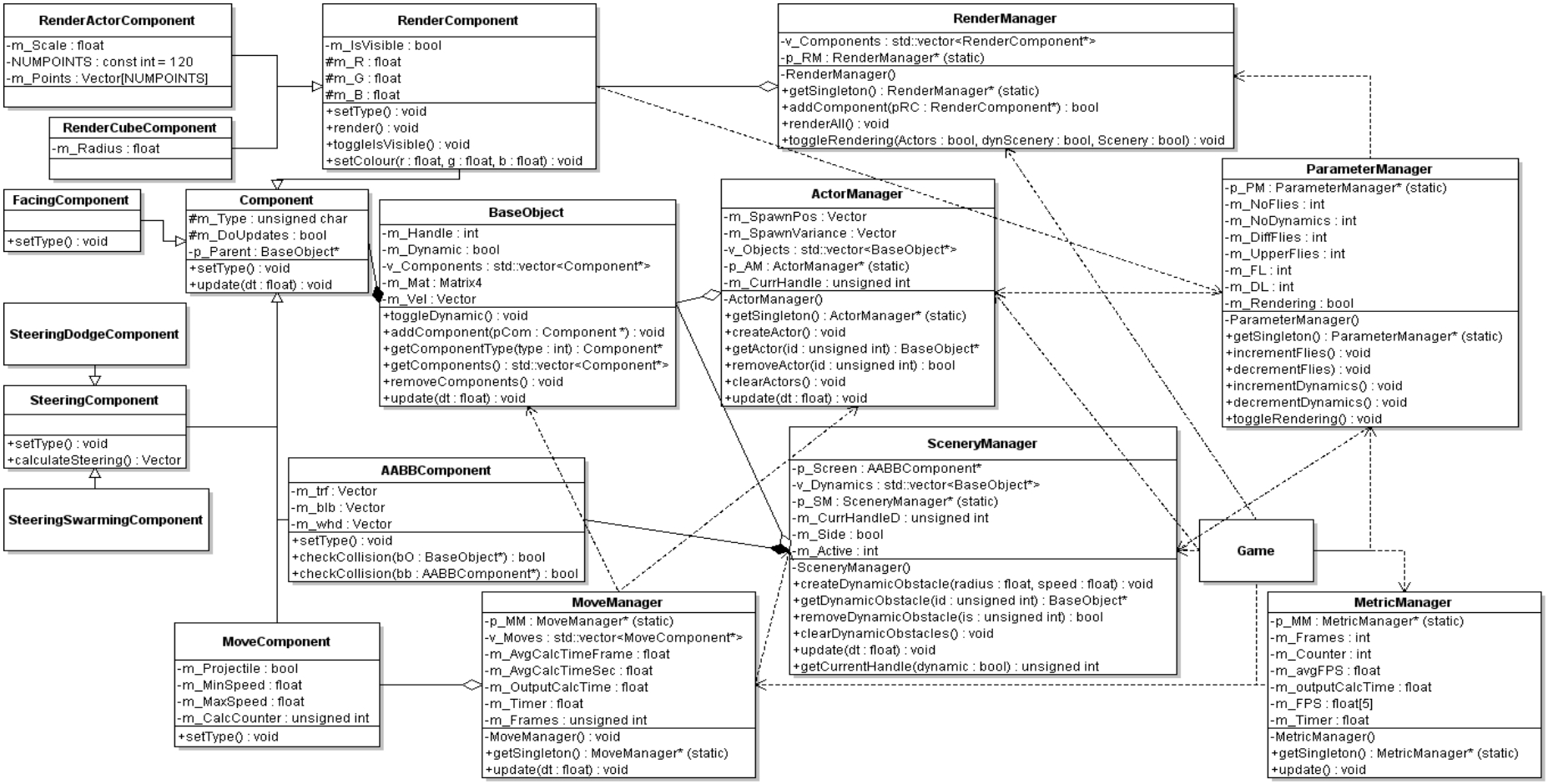


Fig F-1: Final UML Diagram for Implementation Phase 3, Implementation I-1, as implemented in Section 4.3

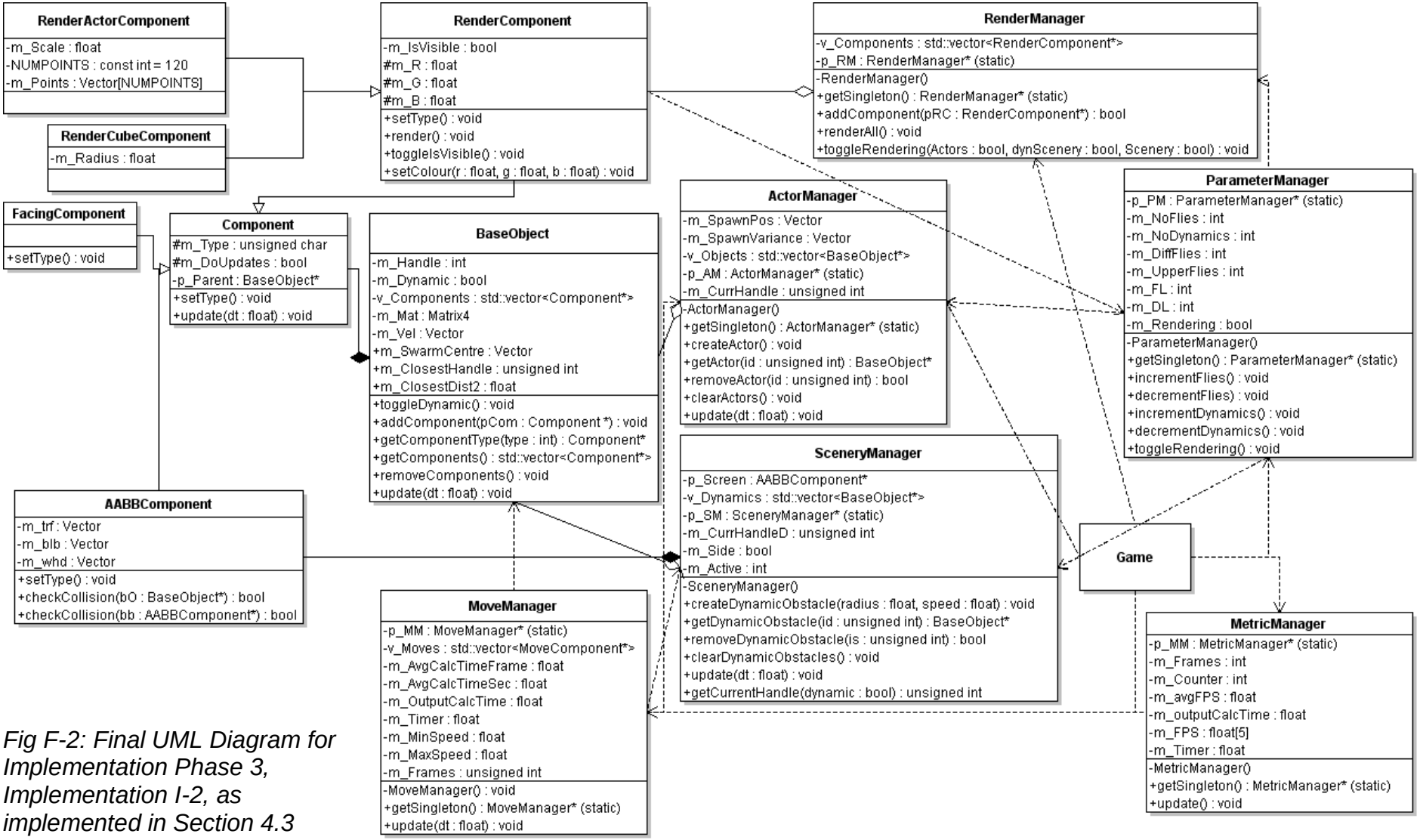


Fig F-2: Final UML Diagram for Implementation Phase 3, Implementation I-2, as implemented in Section 4.3

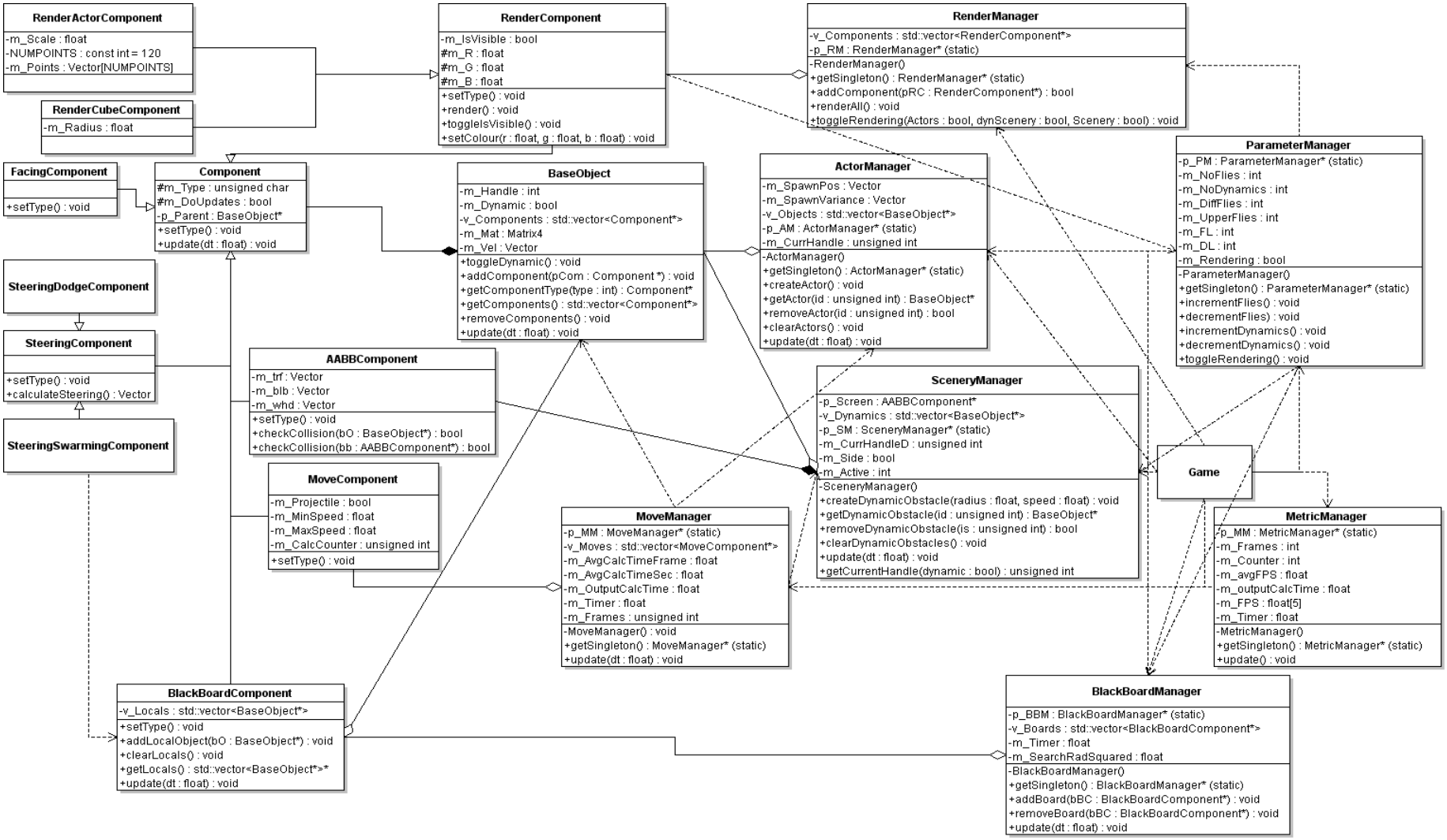


Fig F-3: Final UML Diagram for Implementation Phase 3: Implementation I-3 as implemented in Section 4.3.

Appendix G

The update loop from the Move Manager class, as seen in Implementation Phase 3: Implementation I-2 (reformatted and stripped of comments here), as implemented in Section 4.3.

```
void MoveManager::update(float dt){
 m_timer += dt;
 m_frames++;
 unsigned int counter = 0;

 for(unsigned int i = 0; i < SceneryManager::getSingleton()-
>getCurrentHandle(); i++){
  BaseObject* bO = SceneryManager::getSingleton()-
>getDynamicObstacle(i);
  if(!bO) continue;
  bO->setPosition(bO->getPosition() + (bO->getVelocity() * dt));
 }

 unsigned int time1 = g_Game.getTimer()->timeInMS();
 Vector suggestion, suggestionDodge, suggestionSwarm, flyPos,
checkPos, centre;
 BaseObject* bO, *check;
 for(unsigned int i = 1; i < ActorManager::getSingleton()-
>getCurrentHandle(); i++){
  bO = ActorManager::getSingleton()->getActor(i);
  if(!bO) continue;
  suggestion = suggestionDodge = suggestionSwarm = Vector();
  flyPos = bO->getPosition();
  unsigned int j;

  for(j = 1; j < SceneryManager::getSingleton()->getCurrentHandle();
j++){
   check = SceneryManager::getSingleton()->getDynamicObstacle(j);
   if(!check) continue;
   AABBComponent* bb = (AABBComponent*)check-
>getComponentType(C_TYPE_BOUNDS);
   if(!bb) continue;
   Vector whd = bb->getWHD();
   Vector distance = check->getPosition() - flyPos;

   if(distance.Length2() - ((whd.x * whd.x) + (whd.y * whd.y)) > 256)
continue;
   if(distance.x > 0) distance.x += whd.x;
   else distance.x -= whd.x;
   if(distance.y > 0) distance.y += whd.y;
   else distance.y -= whd.y;

   suggestionDodge -= distance;
  }
  if(suggestionDodge != Vector()) suggestionDodge |= 2.7f;
  counter++;

  for(j = i + 1; j < ActorManager::getSingleton()-
>getCurrentHandle(); j++){
   check = ActorManager::getSingleton()->getActor(j);
   if(!check) continue;
   checkPos = check->getPosition();
```

```
    bO->m_SwarmCentre += checkPos;
    check->m_SwarmCentre += flyPos;

    float d = flyPos.Dist2(checkPos);
    if(d < bO->m_ClosestDist2){
     bO->m_ClosestHandle = j;
     check->m_ClosestHandle = i;

     bO->m_ClosestDist2 = d;
     check->m_ClosestDist2 = d;
    }
   }

   if(j > ActorManager::getSingleton()->getCurrentHandle()) j--;
   bO->m_SwarmCentre /= (float)j;
   Vector mouse(((float)g_Game.m_mouse.x / 3), -
((float)g_Game.m_mouse.y / 3));
   bO->m_SwarmCentre = mouse - bO->m_SwarmCentre;

   if(bO->m_ClosestDist2 > bO->getVelocity().Length2() / 49){
    suggestionSwarm = bO->m_SwarmCentre - flyPos;
    suggestionSwarm |= 1.7f;
   }else{
    BaseObject* dodge = ActorManager::getSingleton()->getActor(bO-
>m_ClosestHandle);
    if(dodge){
     suggestionSwarm = -dodge->getPosition() + flyPos;
     suggestionSwarm |= 1.4f;
    }
   }
   counter++;

   suggestion += bO->getVelocity() +suggestionDodge +suggestionSwarm;
   if(suggestion.Length2() < (m_MinSpeed * m_MinSpeed))
    suggestion |= m_MinSpeed;
   else if(suggestion.Length2() > (m_MaxSpeed * m_MaxSpeed))
    suggestion |= m_MaxSpeed;

   bO->setVelocity(suggestion);
   flyPos += bO->getVelocity() * dt;
   bO->setPosition(flyPos);

   bO->m_SwarmCentre = Vector();
   bO->m_ClosestHandle = -1;
   bO->m_ClosestDist2 = 1000000;
  }
  unsigned int time2 = g_Game.getTimer()->timeInMS();

  m_avgCalcTimeFrame = (float)(time2 - time1) / counter;
  m_avgCalcTimeSec += m_avgCalcTimeFrame;

  if(m_timer < 1) return;

  m_avgCalcTimeSec /= m_frames;
  m_outputCalcTime = m_avgCalcTimeSec;
  m_timer = 0;
  m_frames = 0;
  m_avgCalcTimeSec = 0;
}
```

The calculateSteering function for the Implementation I-3 version of SteeringSwarmingComponent (stripped of comments and reformatted here), as implemented in Section 4.3:

```
Vector SteeringSwarmingComponent::calculateSteering(){
 BlackBoardComponent* BBC = (BlackBoardComponent*)getParent()-
>getComponentType(C_TYPE_BLACKBOARD);
 if(!BBC) return Vector();

 Vector suggestion(0);
 Vector centre(0);
 Vector closest(0);
 unsigned int i;
 float closestdist = 1000.0f;
 std::vector<BaseObject*> v = (*BBC->getLocals());
 BaseObject* check = NULL;

 for(i = 0; i < v.size(); i++){
  check = v[i];
  if(!check) continue;

  centre += check->getPosition();
  Vector c = check->getPosition() - getParent()->getPosition();
  float d = c.Length();

  if(d < closestdist){
   closest = c;
   closestdist = d;
   }
 }

 if(i > 0) centre *= (1.0f / i);

 Vector mouse((float)g_Game.m_mouse.x/3,-(float)g_Game.m_mouse.y/3));
 centre = mouse - centre;

 if(closestdist > getParent()->getVelocity().Length() / 10.0f){
  Vector centredir = centre - getParent()->getPosition();
  centredir.Normalize();
  suggestion += centredir * 1.7f;
 }else{
  closest.Normalize();
  suggestion -= closest * 1.4f;
 }

 return suggestion;
}
```

The update loop for the blackboard manager – makes sure all of the flies are kept up-to-date on what's around them to the nearest update interval time (1 second in the executable on the accompanying CD). From Implementation I-3 (stripped of comments and reformatted here), as implemented in Section 4.3:

```
void BlackBoardManager::update(float dt){
 m_Timer += dt;
 if(m_Timer < UPDATE_INTERVAL) return;

 BlackBoardComponent* BBC = NULL;
 BlackBoardComponent* check = NULL;
 BaseObject* bO, *checkObject;
 Vector pos, checkPos;

 for(unsigned int i = 0; i < v_Boards.size(); i++){
  BBC = v_Boards[i];
  if(!BBC) continue;
  BBC->clearLocals();
 }

 for(unsigned int i = 0; i < v_Boards.size(); i++){
  BBC = v_Boards[i];
  if(!BBC) continue;
  bO = BBC->getParent();
  if(!bO) continue;
  pos = bO->getPosition();

  for(unsigned int j = i + 1; j < v_Boards.size(); j++){
   check = v_Boards[j];
   if(!check) continue;
   checkObject = check->getParent();
   if(!checkObject) continue;
   checkPos = checkObject->getPosition();

   if(checkPos.Dist2(pos) > m_SearchRadSquared) continue;

   BBC->addLocalObject(checkObject);
   check->addLocalObject(bO);
  }
 }

 m_Timer = 0;
}
```

Appendix H

Results tables for Section 5.1 – these results represent the recorded, official testing for the project in order to achieve the goal of perfecting the effectiveness of the swarming behaviour through parameter values.

Table H-1: Results for testing Minimum and Maximum Speeds with both behaviour weightings set to 1.0. A minimum range of 1.0 was kept between minimum and maximum speeds. Cohesion and Separation weightings were set to 1.0, and 40 flies were used in this test.

Speed		Average Speed	Average Distance to Centre	Avg. Speed within Avg. Distance	Avg. Speed out of Avg. Distance	Appearance Rating (out of 10)
Min	Max					
10	11	10.5	5.2	10.3	10.7	5
	12	10.8	6.4	11.1	10.6	5
	13	11.5	5.6	11.5	11.4	5
	14	12.1	5.3	12.1	12.0	5
	15	12.4	5.7	12.6	11.9	5
	16	13.1	6.2	12.8	13.3	6
	17	13.2	7.2	13.5	12.8	6
	18	14.4	7.3	14.2	14.7	6
	19	14.5	8.1	13.8	15.3	7
	20	16.1	8.7	16.7	15.6	6
11	12	11.5	5.8	11.5	11.4	5
	13	11.9	6.5	12.2	11.7	6
	14	12.5	6.2	12.7	12.3	7
	15	13.1	6.4	13.4	12.8	7
	16	13.3	8.7	14.1	12.4	7
	17	13.5	8.2	14.1	12.9	8
	18	14.3	8.4	14.3	14.3	8
	19	14.5	7.9	14.3	14.7	7
	20	15.7	7.4	16.0	15.2	7
12	13	12.5	6.8	12.6	12.5	6
	14	12.8	6.9	12.9	12.7	7
	15	13.9	6.5	14.0	13.7	8
	16	13.8	7.1	13.9	13.6	7
	17	14.8	7.2	14.8	14.8	8
	18	14.9	9.4	15.0	14.7	8
	19	16.3	8.8	15.9	16.6	7
	20	16.0	8.6	16.1	15.9	7
13	14	13.4	7.3	13.5	13.3	6
	15	14.1	8.8	14.0	14.2	8
	16	14.1	7.8	14.1	14.2	7
	17	15.0	8.5	15.9	13.8	8
	18	16.0	9.1	16.2	15.9	7
	19	16.4	8.3	16.7	16.0	7
	20	16.7	8.7	17.3	15.9	7

14	15	14.5	9.6	14.5	14.4	7
	16	15.1	7.8	15.0	15.2	8
	17	15.6	8.1	15.6	15.6	8
	18	15.5	9.2	15.5	15.6	8
	19	16.4	9.1	16.7	16.1	7
	20	17.0	9.1	17.0	17.0	6
15	16	15.5	10.2	15.5	15.6	6
	17	16.1	10.0	15.9	16.3	6
	18	16.4	9.0	16.6	16.3	6
	19	17.1	10.1	17.2	16.9	6
	20	17.5	11.4	17.1	17.9	6
16	17	16.4	13.6	16.5	16.3	5
	18	17.2	11.6	17.1	17.3	5
	19	17.3	11.0	17.2	17.5	6
	20	18.0	13.0	18.5	17.3	6
17	18	17.5	13.0	17.5	17.4	4
	19	17.8	11.9	17.9	17.7	4
	20	18.9	11.2	19.2	18.6	5
18	19	18.4	14.8	18.4	18.4	3
	20	19.0	13.7	19.0	19.0	4
19	20	19.4	13.3	19.5	19.4	2

Table H-2: Results for testing the weightings of Separation and Cohesion behaviours, with the minimum speed set to 12 and the maximum speed at 18. The values are tested against each other to better determine an effective combination.

Separation Weighting	Cohesion Weighting	Average Speed	Average Distance to Centre	Avg. Speed within Avg. Distance	Avg. Speed out of Avg. Distance	Appearance Rating (out of 10)
0.1	3.0	16.3	6.8	17.2	14.8	5
0.2	2.9	16.5	5.8	16.7	16.2	5
0.3	2.8	16.2	6.0	16.6	15.8	5
0.4	2.7	15.2	6.1	15.5	14.9	5
0.5	2.6	15.5	5.8	16.2	14.8	5
0.6	2.5	15.7	5.7	15.6	15.9	5
0.7	2.4	15.1	5.2	15.0	15.2	6
0.8	2.3	14.3	6.8	15.5	13.3	6
0.9	2.2	15.3	6.6	15.4	15.2	7
1.0	2.1	14.9	6.5	15.2	14.5	7
1.1	2.0	15.6	6.0	15.6	15.6	7
1.2	1.9	14.9	6.5	14.7	15.0	8
1.3	1.8	14.4	6.0	14.6	14.3	8
1.4	1.7	14.9	6.6	15.1	14.8	8
1.5	1.6	15.5	6.4	15.3	15.8	8
1.6	1.5	15.9	6.3	16.1	15.4	8
1.7	1.4	14.5	7.4	14.1	14.9	7
1.8	1.3	15.1	6.1	15.7	14.2	7
1.9	1.2	14.6	6.1	14.5	14.6	8
2.0	1.1	15.1	6.9	15.4	14.8	7
2.1	1.0	14.0	8.6	13.5	14.7	7
2.2	0.9	15.2	9.9	16.0	13.9	6
2.3	0.8	15.8	8.8	15.4	16.4	7
2.4	0.7	13.8	12.2	14.6	12.9	6
2.5	0.6	14.3	11.5	14.9	13.5	6
2.6	0.5	14.4	11.6	14.7	14.1	6
2.7	0.4	14.4	16.3	15.0	13.0	4
2.8	0.3	14.6	17.4	15.3	13.2	3
2.9	0.2	14.1	22.7	14.3	13.8	2
3.0	0.1	13.1	43.8	13.7	12.3	1

Table H-3: Listing of parameter values deemed most effective in combination, with the metric results and appearance rating as proof of choice. This table shows a small range of values that seemed most promising, compared against each other as 9 different combinations to determine an ideal behaviour.

Minimum Speed	12	12	12	13	13	13	14	14	14
Maximum Speed	17	17	18	17	17	18	17	18	18
Separation Weighting	1.2	1.3	1.4	1.2	1.3	1.4	1.2	1.3	1.4
Cohesion Weighting	1.9	1.8	1.7	1.8	1.7	1.9	1.7	1.9	1.8
Average Speed	14.5	14.4	15.3	14.6	15.4	14.8	15.4	16.1	16.1
Average Distance to Centre	7.5	6.4	6.4	7.3	7.0	7.4	7.1	7.5	7.2
Avg. Speed Within Avg. Distance	14.9	14.4	15.3	14.7	14.8	14.9	15.7	16.3	16.0
Avg. Speed out of Avg. Distance	14.0	14.3	15.3	14.5	15.9	14.8	15.0	15.8	16.4
Appearance Rating (out of 10)	8	8	9	8	9	9	9	9	9

Table H-4: Results of the scaling investigation - the number of flies in the swarm was altered for the selected parameter values (Minimum Speed: 14, Maximum Speed: 18, Cohesion Weighting: 1.7, Separation Weighting 1.4).

Number of Flies in Swarm	Average Speed	Average Distance to Centre	Avg. Speed Within Avg. Distance	Avg. Speed out of Avg. Distance	Appearance Rating (out of 10)
40	16.1	6.6	16.3	15.8	10
80	15.6	8.1	15.3	16.0	10
120	15.6	9.9	16.1	15.2	10
160	15.8	11.1	15.6	16.2	10
200	15.4	12.2	15.2	15.7	10
240	15.6	12.0	15.7	15.5	8
280	15.6	12.7	15.7	15.6	9
320	15.5	13.7	15.6	15.4	10
360	15.4	14.0	15.4	15.5	9
400	15.7	15.3	15.7	15.6	9

Appendix I

Results tables for Section 5.2 – these results are all from the official, recorded testing of the implementation of dynamic obstacle avoidance behaviour from Section 4.2. This testing was done in order to find the best parameter values for making the behaviour effective in reducing the number of collisions between flies and dynamic obstacles to as near to zero as possible, whilst maintaining an acceptable appearance.

Table I-1: Results for testing distance at which 60 flies become aware of a dynamic obstacle and weighting of avoidance behaviour against number of collisions. Appearance is also rated subjectively, and there are always two objects that are 10 square units in size moving at a speed of 12 units.

Avoidance Behaviour Weighting	Distance of Object-Awareness	Number of Collisions	Appearance Rating (out of 10)
0.1	4	89	0
	8	87	0
	12	87	0
	16	90	0
	20	76	0
	24	80	0
	28	83	0
	32	75	0
0.2	4	89	0
	8	76	0
	12	87	0
	16	83	0
	20	76	0
	24	69	0
	28	76	0
	32	85	0
0.3	4	87	0
	8	72	0
	12	67	0
	16	79	0
	20	79	0
	24	69	0
	28	88	0
	32	78	0
0.4	4	91	0
	8	78	0
	12	80	0
	16	72	0
	20	69	0
	24	73	0
	28	77	0
	32	81	0

0.5	4	68	0
	8	71	0
	12	72	0
	16	62	0
	20	68	0
	24	71	0
	28	75	0
	32	78	0
0.6	4	71	0
	8	56	1
	12	51	1
	16	57	1
	20	67	0
	24	58	0
	28	76	0
	32	77	0
0.7	4	60	0
	8	49	2
	12	63	1
	16	54	2
	20	51	2
	24	64	0
	28	60	0
	32	70	0
0.8	4	55	1
	8	36	4
	12	35	4
	16	56	3
	20	38	4
	24	58	1
	28	69	0
	32	70	0
0.9	4	50	3
	8	43	4
	12	40	4
	16	27	5
	20	37	4
	24	36	4
	28	58	1
	32	52	1
1.0	4	61	1
	8	28	5
	12	27	5
	16	22	5
	20	38	5
	24	33	4
	28	52	2
	32	31	4

1.1	4	42	3
	8	27	4
	12	34	4
	16	25	5
	20	30	5
	24	30	4
	28	36	4
	32	30	3
1.2	4	33	4
	8	32	4
	12	30	5
	16	31	5
	20	34	4
	24	16	4
	28	28	3
	32	40	3
1.3	4	25	4
	8	25	5
	12	21	5
	16	9	5
	20	15	4
	24	18	4
	28	16	4
	32	20	3
1.4	4	20	4
	8	25	5
	12	11	6
	16	20	6
	20	7	5
	24	17	4
	28	6	4
	32	15	3
1.5	4	29	4
	8	17	6
	12	15	7
	16	10	6
	20	7	6
	24	11	5
	28	9	4
	32	10	3
1.6	4	19	4
	8	18	6
	12	11	7
	16	14	7
	20	4	6
	24	6	4
	28	5	4
	32	3	3

1.7	4	17	5
	8	14	7
	12	7	7
	16	3	7
	20	2	6
	24	3	4
	28	4	4
	32	3	3
1.8	4	19	5
	8	14	7
	12	9	8
	16	3	7
	20	1	6
	24	1	4
	28	2	4
	32	2	3
1.9	4	17	5
	8	7	8
	12	4	8
	16	0	8
	20	1	7
	24	4	5
	28	1	3
	32	0	3
2.0	4	16	5
	8	10	8
	12	1	8
	16	4	8
	20	1	7
	24	4	5
	28	1	3
	32	1	2
2.1	4	20	5
	8	7	8
	12	3	10
	16	3	9
	20	1	7
	24	1	5
	28	0	3
	32	0	2
2.2	4	19	5
	8	6	9
	12	4	10
	16	1	10
	20	0	6
	24	2	5
	28	0	3
	32	0	2

2.3	4	9	5
	8	4	9
	12	0	10
	16	2	10
	20	1	6
	24	0	4
	28	0	2
	32	0	1
2.4	4	17	5
	8	4	9
	12	0	10
	16	0	10
	20	3	6
	24	0	5
	28	1	3
	32	0	2
2.5	4	8	5
	8	3	9
	12	0	10
	16	0	10
	20	0	6
	24	0	4
	28	0	2
	32	0	1
2.6	4	5	4
	8	1	8
	12	0	8
	16	0	7
	20	0	5
	24	0	3
	28	0	1
	32	0	0
2.7	4	9	4
	8	1	7
	12	0	8
	16	0	7
	20	1	5
	24	0	4
	28	0	2
	32	0	0
2.8	4	6	4
	8	0	7
	12	2	6
	16	0	6
	20	0	4
	24	0	2
	28	0	0
	32	0	0

2.9	4	4	3
	8	0	5
	12	0	4
	16	0	4
	20	0	3
	24	0	1
	28	0	0
	32	0	0
3.0	4	4	3
	8	2	3
	12	0	4
	16	0	4
	20	0	3
	24	0	2
	28	0	0
	32	0	0
3.1	4	7	2
	8	4	3
	12	0	2
	16	0	2
	20	0	2
	24	0	1
	28	0	0
	32	0	0
3.2	4	6	2
	8	2	3
	12	0	2
	16	0	1
	20	0	1
	24	0	0
	28	0	0
	32	0	0
3.3	4	4	1
	8	1	2
	12	0	1
	16	0	1
	20	0	0
	24	0	0
	28	0	0
	32	0	0
3.4	4	6	1
	8	2	1
	12	1	1
	16	0	0
	20	0	0
	24	0	0
	28	0	0
	32	0	0

3.5	4	5	1
	8	0	1
	12	0	0
	16	0	0
	20	0	0
	24	0	0
	28	0	0
	32	0	0
3.6	4	0	0
	8	0	0
	12	0	0
	16	0	0
	20	0	0
	24	0	0
	28	0	0
	32	0	0
3.7	4	0	0
	8	0	0
	12	0	0
	16	0	0
	20	0	0
	24	0	0
	28	0	0
	32	0	0
3.8	4	3	0
	8	0	0
	12	0	0
	16	0	0
	20	0	0
	24	0	0
	28	0	0
	32	0	0
3.9	4	0	0
	8	0	0
	12	0	0
	16	0	0
	20	0	0
	24	0	0
	28	0	0
	32	0	0
4.0	4	1	0
	8	0	0
	12	0	0
	16	0	0
	20	0	0
	24	0	0
	28	0	0
	32	0	0

Table I-2: Results for the selected weighting (2.5) and distance (12) against a varying number of shapes with identical size (10) and speed (12).

Number of Shapes	Number of Collisions	Appearance Rating (out of 10)
1	1	10
2	5	9
3	3	10
4	6	10
5	12	9

Table I-3: Results for the selected weighting (boosted to 2.7) in the same test as above.

Number of Shapes	Number of Collisions	Appearance Rating (out of 10)
1	0	10
2	1	10
3	2	10
4	4	10
5	5	10

Table I-4: Results for the selected weighting (2.7) and distance (12) against 2 shapes of varying size and identical speed (12) values.

Radius of Shapes	Number of Collisions	Appearance Rating (out of 10)
2	0	9
4	0	9
6	4	9
8	3	10
10	0	10
12	1	10
14	1	10
16	3	10
18	3	10
20	3	10

Table I-5: Results for the selected parameter values against 2 shapes of identical size and varying speed values.

Speed of Shapes	Number of Collisions	Appearance Rating (out of 10)
3	1	9
6	1	9
9	3	10
12	3	10
15	0	10
18	5	10
21	22	10
24	26	10
27	31	10
30	63	8

Appendix J

Results tables for Section 5.3 – these results are from the official, recorded efficiency testing of the three implementations detailed in Section 4.3. The implementations are labelled as such:

- I-1 – Standard implementation as used for previous stages of testing.
- I-2 – Trimmed and efficiency-minded implementation.
- I-3 – Blackboard-based implementation.

Table J-1: Results for efficiency testing with rendering on.

Number of Flies	Number of Obstacles	Average FPS (over last 5 seconds)			Average Behaviour Calculation Time per Object per Frame (ms)		
		I-1	I-2	I-3	I-1	I-2	I-3
50	**0**	**59.40**	**59.40**	**58.64**	**0.0025**	**0.0000**	**0.0027**
	1	**59.21**	**59.60**	**56.48**	**0.0053**	**0.0000**	**0.0026**
	2	**59.20**	**59.61**	**58.61**	**0.0053**	**0.0027**	**0.0000**
	3	**59.44**	**59.60**	**58.43**	**0.0051**	**0.0025**	**0.0025**
	4	**59.30**	**59.43**	**56.84**	**0.0027**	**0.0000**	**0.0028**
	5	**59.41**	**59.60**	**58.80**	**0.0025**	**0.0027**	**0.0026**
100	0	59.40	59.60	58.43	0.0040	0.0014	0.0025
	1	59.60	59.24	58.20	0.0038	0.0013	0.0039
	2	59.42	59.04	58.60	0.0041	0.0013	0.0040
	3	59.60	59.40	58.00	0.0052	0.0000	0.0013
	4	59.40	59.40	57.64	0.0054	0.0000	0.0055
	5	59.20	59.61	59.40	0.0040	0.0000	0.0052
150	**0**	**59.40**	**59.20**	**58.20**	**0.0061**	**0.0009**	**0.0018**
	1	**59.20**	**59.04**	**59.00**	**0.0071**	**0.0017**	**0.0026**
	2	**59.03**	**59.41**	**58.46**	**0.0071**	**0.0017**	**0.0026**
	3	**59.20**	**59.20**	**58.60**	**0.0061**	**0.0018**	**0.0035**
	4	**59.63**	**59.61**	**58.08**	**0.0061**	**0.0008**	**0.0036**
	5	**59.62**	**59.24**	**58.21**	**0.0052**	**0.0009**	**0.0035**
200	0	59.20	59.23	57.63	0.0081	0.0020	0.0041
	1	59.43	58.40	56.85	0.0097	0.0014	0.0049
	2	59.21	59.00	57.80	0.0113	0.0019	0.0035
	3	59.40	59.60	58.60	0.0090	0.0020	0.0041
	4	59.40	59.22	56.20	0.0093	0.0006	0.0036
	5	59.40	59.20	57.83	0.0099	0.0013	0.0027
250	**0**	**58.85**	**59.20**	**58.00**	**0.0099**	**0.0026**	**0.0032**
	1	**59.20**	**59.00**	**55.60**	**0.0095**	**0.0011**	**0.0034**
	2	**59.20**	**59.40**	**55.60**	**0.0101**	**0.0016**	**0.0050**
	3	**59.61**	**59.60**	**56.63**	**0.0101**	**0.0026**	**0.0049**
	4	**59.20**	**59.22**	**58.22**	**0.0112**	**0.0026**	**0.0052**
	5	**59.40**	**59.60**	**57.20**	**0.0094**	**0.0021**	**0.0033**

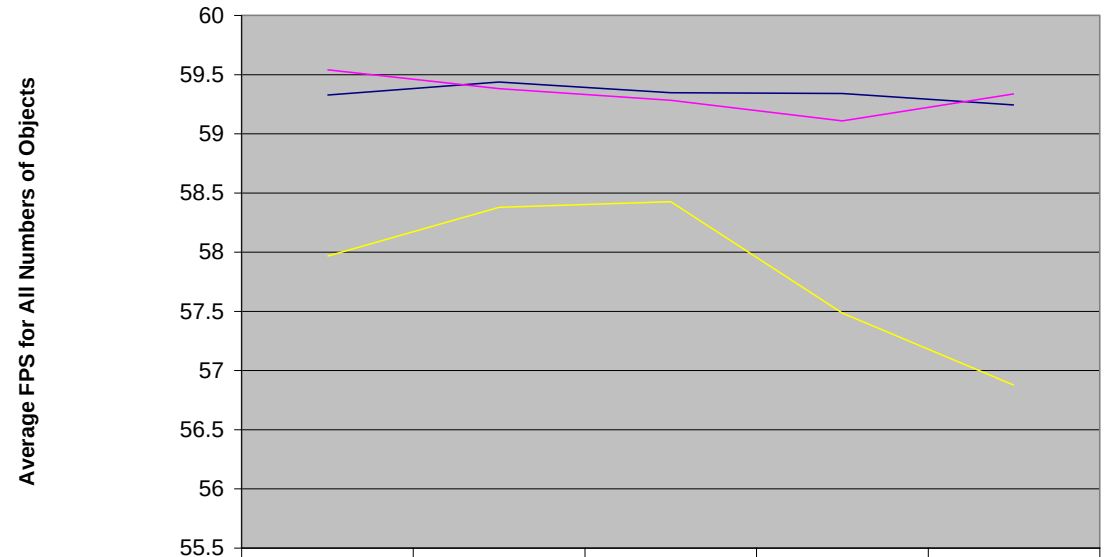
Table J-2: Results for efficiency testing with rendering off.

Number of Flies	Number of Obstacles	Average FPS (over last 5 seconds)			Average Behaviour Calculation Time per Object per Frame (ms)		
		I-1	I-2	I-3	I-1	I-2	I-3
100	**0**	**59.25**	**59.56**	**58.44**	**0.0039**	**0.0011**	**0.0027**
	1	**59.60**	**59.55**	**58.47**	**0.0039**	**0.0010**	**0.0013**
	2	**58.63**	**59.55**	**57.48**	**0.0039**	**0.0015**	**0.0026**
	3	**58.66**	**59.56**	**58.00**	**0.0027**	**0.0006**	**0.0014**
	4	**59.23**	**59.55**	**58.20**	**0.0026**	**0.0010**	**0.0013**
	5	**59.00**	**59.58**	**59.22**	**0.0039**	**0.0011**	**0.0040**
200	0	59.24	59.55	58.80	0.0083	0.0013	0.0033
	1	59.20	59.54	58.43	0.0086	0.0012	0.0027
	2	58.60	59.52	58.02	0.0108	0.0013	0.0026
	3	59.04	59.55	58.45	0.0101	0.0015	0.0020
	4	59.42	59.55	59.01	0.0080	0.0012	0.0033
	5	58.84	59.58	57.03	0.0091	0.0015	0.0040
300	**0**	**59.20**	**59.60**	**59.22**	**0.0135**	**0.0018**	**0.0043**
	1	**59.40**	**58.81**	**58.00**	**0.0121**	**0.0026**	**0.0049**
	2	**58.80**	**59.21**	**58.62**	**0.0142**	**0.0026**	**0.0056**
	3	**59.00**	**59.22**	**58.21**	**0.0134**	**0.0018**	**0.0023**
	4	**59.61**	**59.20**	**58.80**	**0.0152**	**0.0027**	**0.0044**
	5	**58.82**	**59.40**	**58.74**	**0.0131**	**0.0022**	**0.0049**
400	0	58.20	59.41	58.63	0.0165	0.0036	0.0049
	1	57.20	59.42	58.20	0.0164	0.0033	0.0034
	2	58.40	59.40	57.63	0.0182	0.0029	0.0050
	3	57.83	59.20	57.83	0.0172	0.0026	0.0041
	4	57.21	59.42	58.80	0.0152	0.0030	0.0040
	5	58.42	59.00	58.40	0.0173	0.0027	0.0039
500	**0**	**41.61**	**59.60**	**58.20**	**0.0220**	**0.0034**	**0.0054**
	1	**41.27**	**59.20**	**58.83**	**0.0206**	**0.0042**	**0.0060**
	2	**41.02**	**59.62**	**58.41**	**0.0213**	**0.0044**	**0.0042**
	3	**40.87**	**58.83**	**57.64**	**0.0214**	**0.0040**	**0.0058**
	4	**40.80**	**59.20**	**58.60**	**0.0233**	**0.0037**	**0.0072**
	5	**40.02**	**59.60**	**58.21**	**0.0209**	**0.0050**	**0.0092**
600	0	29.83	59.63	58.40	0.0256	0.0043	0.0057
	1	29.94	59.60	58.82	0.0256	0.0050	0.0056
	2	29.43	59.41	58.24	0.0261	0.0049	0.0059
	3	29.62	59.40	58.04	0.0251	0.0040	0.0098
	4	29.32	59.41	58.00	0.0256	0.0052	0.0089
	5	29.23	59.40	56.82	0.0256	0.0057	0.0093
700	**0**	**22.38**	**59.40**	**58.20**	**0.0291**	**0.0050**	**0.0042**
	1	**22.39**	**59.60**	**57.85**	**0.0296**	**0.0049**	**0.0057**
	2	**22.44**	**59.60**	**57.43**	**0.0291**	**0.0057**	**0.0070**
	3	**22.32**	**59.40**	**58.22**	**0.0300**	**0.0051**	**0.0054**
	4	**22.38**	**59.20**	**59.21**	**0.0296**	**0.0052**	**0.0088**
	5	**21.72**	**59.60**	**58.61**	**0.0300**	**0.0061**	**0.0067**

800	0	17.51	59.40	57.80	0.0341	0.0057	0.0064
	1	17.51	59.40	58.64	0.0331	0.0065	0.0063
	2	17.56	59.02	57.25	0.0342	0.0063	0.0063
	3	17.46	59.60	57.00	0.0347	0.0060	0.0084
	4	17.51	59.00	56.45	0.0342	0.0066	0.0085
	5	17.40	58.80	57.00	0.0336	0.0062	0.0074
900	**0**	**14.05**	**59.01**	**57.40**	**0.0382**	**0.0079**	**0.0078**
	1	**14.07**	**59.40**	**57.41**	**0.0382**	**0.0068**	**0.0078**
	2	**14.02**	**59.20**	**55.85**	**0.0384**	**0.0069**	**0.0082**
	3	**14.00**	**59.22**	**57.03**	**0.0378**	**0.0069**	**0.0084**
	4	**13.96**	**59.00**	**55.20**	**0.0385**	**0.0074**	**0.0083**
	5	**13.65**	**58.80**	**50.20**	**0.0388**	**0.0070**	**0.0090**
1000	0	11.46	52.51	54.20	0.0411	0.0080	0.0062
	1	11.39	52.47	52.24	0.0423	0.0078	0.0072
	2	11.50	52.00	53.84	0.0410	0.0078	0.0077
	3	11.40	51.87	44.19	0.0417	0.0078	0.0109
	4	11.36	52.00	50.66	0.0423	0.0081	0.0059
	5	11.39	51.80	49.27	0.0423	0.0079	0.0102

Appendix K

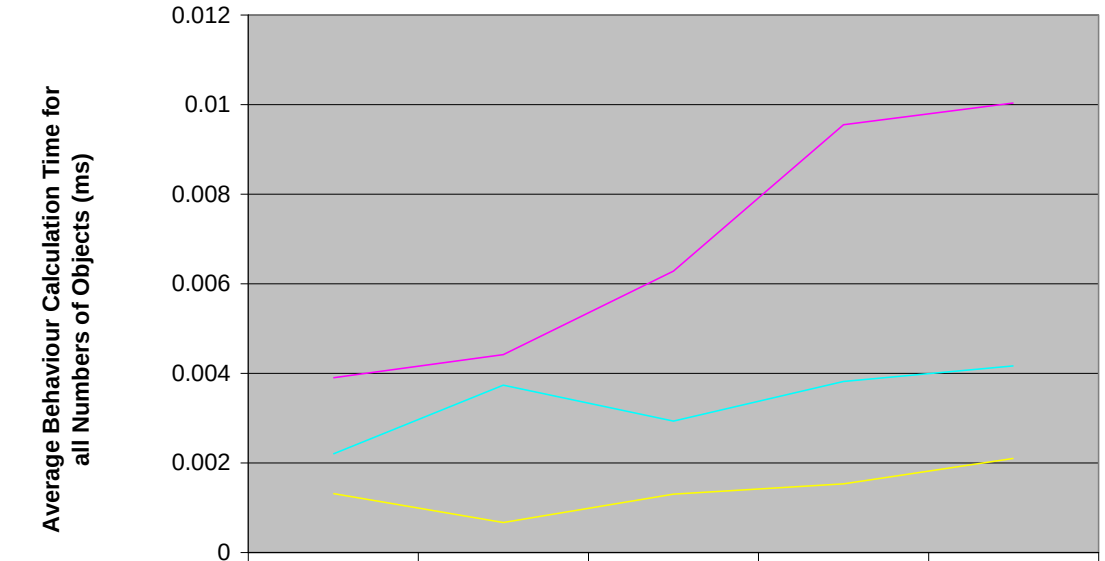
All tables for data from Appendix J.



	50	100	150	200	250
Implementation I-1	59.32666667	59.43666667	59.34666667	59.34	59.24333333
Implementation I-2	59.54	59.38166667	59.28333333	59.10833333	59.33666667
Implementation I-3	57.96666667	58.37833333	58.425	57.485	56.875

Number of Flies

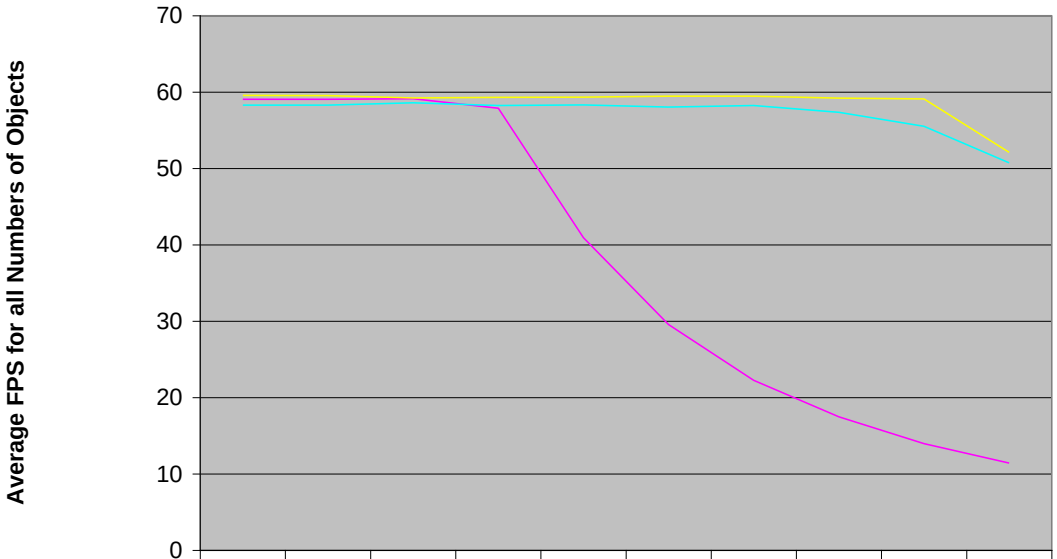
Table K-1: *Average FPS for all Numbers of Objects over Number of Flies (Rendering on)*



	50	100	150	200	250
Implementation I-1	0.0039	0.004416667	0.006283333	0.00955	0.010033333
Implementation I-2	0.001316667	0.000666667	0.0013	0.001533333	0.0021
Implementation I-3	0.0022	0.003733333	0.002933333	0.003816667	0.004166667

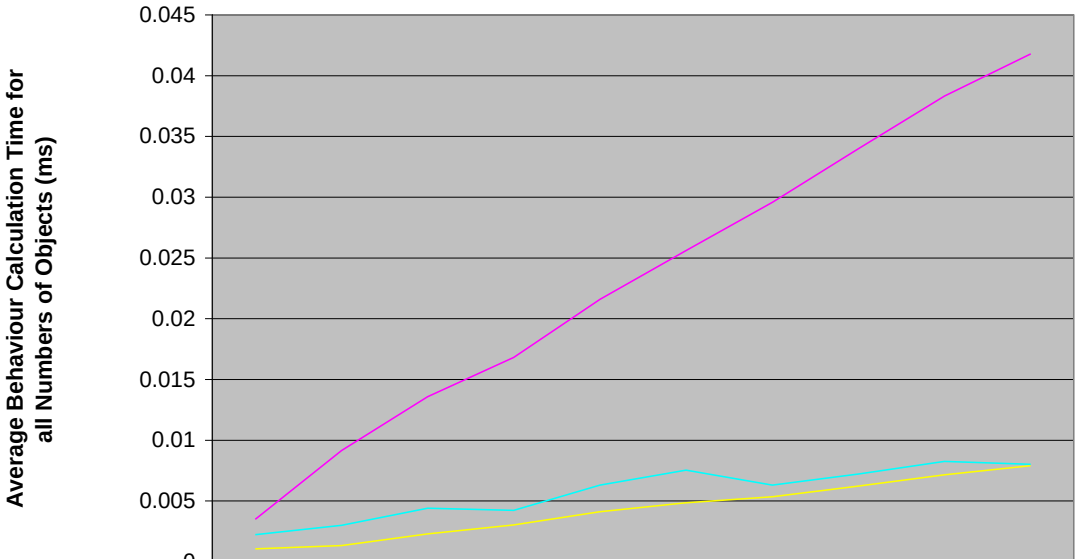
Number of Flies

Table K-2: *Average Behaviour Calculation Time for all Numbers of Objects over Number of Flies (Rendering on)*



	100	200	300	400	500	600	700	800	900	1000
Implementation I-1	59.062	59.057	59.138	57.877	40.932	29.562	22.272	17.492	13.958	11.417
Implementation I-2	59.558	59.548	59.24	59.308	59.342	59.475	59.467	59.203	59.105	52.108
Implementation I-3	58.302	58.29	58.598	58.248	58.315	58.053	58.253	57.357	55.515	50.733

Table K-3: *Average FPS for all Numbers of Objects over Number of Flies (Rendering off)*



	100	200	300	400	500	600	700	800	900	1000
Implementation I-1	0.0035	0.0092	0.0136	0.0168	0.0216	0.0256	0.0296	0.034	0.0383	0.0418
Implementation I-2	0.0011	0.0013	0.0023	0.003	0.0041	0.0049	0.0053	0.0062	0.0072	0.0079
Implementation I-3	0.0022	0.003	0.0044	0.0042	0.0063	0.0075	0.0063	0.0072	0.0083	0.008

Table K-4: *Average Behaviour Calculation Time for all Numbers of Objects over Number of Flies (Rendering off)*