Real Time Crowd Simulation

Peter Maloney[1]

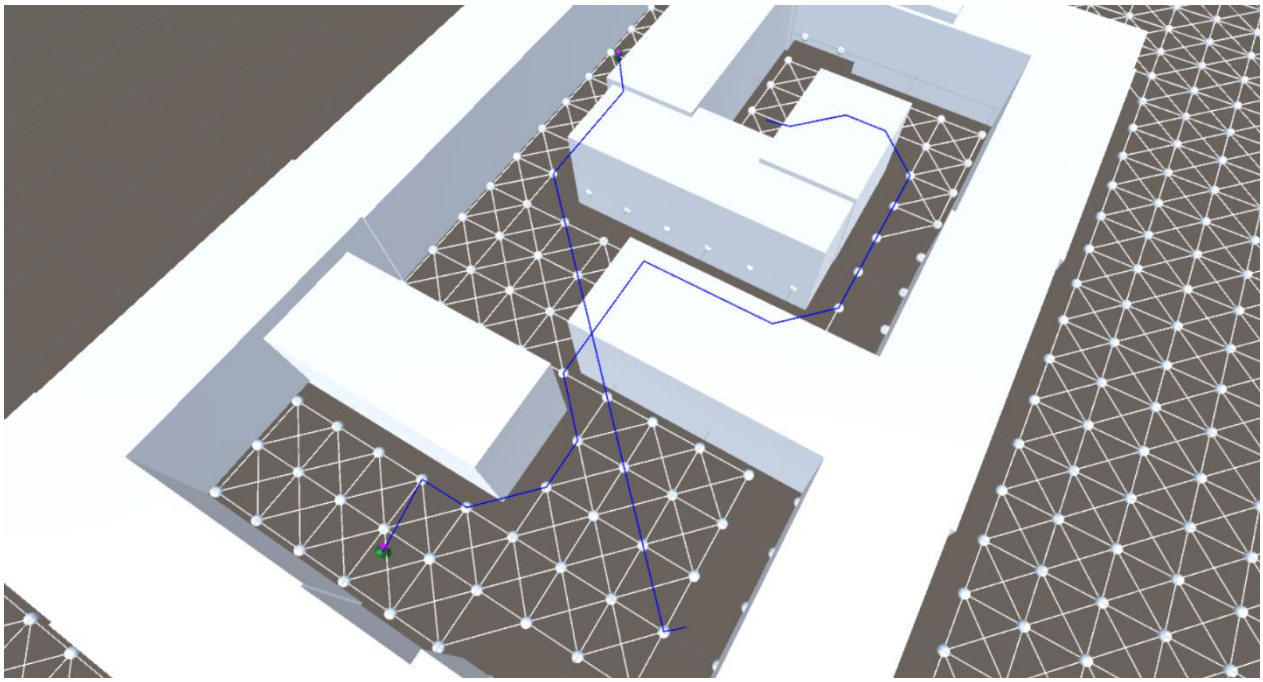
13002995

Supervisor: Dr Simon Scarle

Submission Date 14/04/2016

Module: UFCFS4-30-3 Creative Technologies Project

[1] Department of Computer Science and Creative Technology.
University of the West of England
Coldharbour Lane
Bristol, UK
Peter2.maloney@live.uwe.ac.uk



1 Summary

The final output of the project in question is a basic system comprising of two main components, which combine to simulate crowd behaviors, both autonomously and guided by user input. The system in question contains much of the functionality that was initially proposed, but lacks a large amount of stability and usability that would be expected.

The first main component of the system is a pathfinding sub-system, which utilises *Dijkstra's Algorithm* (Dijkstra, 1959), or variations thereof, to calculate the shortest distance route between two points in a grid of user defined nodes. This system itself makes use of components which detect connections between the nodes, to allow large grids to be easily set up for use. This sub-system outputs a list of the positions of the nodes on the shortest path, which is then used by another component, inspired by *Left 4 Dead's* AI system (The AI Systems of Left 4 Dead), to 'smooth' the often jagged paths created due to the nature of the node grid. This component aims to smooth the path by looking ahead to the furthest visible point on the path, and adjusting agent movement to focus on this, cutting out many of the sharp turns of the un-touched paths.

The second sub-system is a collision avoidance system, which aims to produce more realistic behaviors of agents within the system. While avoidance of larger, static objects is dealt with by cutting node connections through said objects, this sub-system deals with avoidance of smaller, dynamics objects, such as other agents. This sub-system is again inspired by *Left 4 Dead's* AI System (The AI Systems of Left 4 Dead), making use of two colliders in front of, and off to either side of each agent. When either of these colliders detects a collision, it will send a message to the agent, causing it to turn away from the collision, producing, in most cases, smooth and somewhat realistic behaviours.

2 Biography

The developer of this project is a final year Games Technology student at the time of writing, with experience with C++, Unity, and aspects of artificial intelligence as a result of studies. The developer takes a personal interest in video games, and their systems, particularly real time strategy games. By undertaking this project, they hope to develop a system that could be used for future projects, as well as to gain a better understanding of artificial intelligence systems and the potential uses for said systems.

3 How to Access The Project

The entire project can be accessed at: https://github.com/phatpedro21/CreativeTech

The project comprises of One Unity project, titled 'CreativeTechGithub, which will run with Unity Version 5 (older versions may be suitable). The latest version of the project found in the folder titled 'A_Presentations', is made up of Two example scenes:

- The scene titled 'Sandbox' is set up to show off all the components of the system, and was the scene used to produce the examples seen throughout this report as well as the demonstration video.
- The scene titled 'General Showcase' is intended to show the system being applied in a beautified scenario, and should produce behaviors without any user input, by simply 'playing' the scene.

Within either of these scenes, the arrow keys will pan the camera and the mouse wheel with zoom the camera in and out. Left clicking on an agent, will select it and right clicking anywhere with an agent selected will order the agent to move to that point. IT may be necessary to press the 'G' key to make agents move, if they do not by default.

There are also a number of options available in the inspector. On any object with the Pathfinding Showcase script these are:

- A check box for 'Check Distance', checking this will cause pathfinding to use an A* approach, unchecking will use Dijkstra's Algorithm.

- A check box for 'Yield Yes', checking this will spread pathfinding over multiple updates, allowing other actions to be carried out while a path is found, unchecking this will perform pathfinding in one frame, which may cause the system to 'hang'.

- A check box for 'Show Checked Nodes', checking this will highlight each node as it is checked by the pathfinding system.

On any object with the Agent Showcase script the available option is :

- A check box for 'Smooth Paths', checking this will draw lines showing the actual path taken by agents, that is the 'smoothed' path as opposed to the exact path created by the pathfinding system.

4 Introduction

This project set out to produce a system that could be utilised to primarily as a tool for implementing realistic crowd behaviors within video games, specifically RTS (Real Time Strategy) games, but a guiding factor of development to was to ensure the system was as versatile as possible, opening up potential uses to include the creation of CGI crowds in film or building realistic simulations for training, to name a few examples.

Alongside the production of a useful tool, this project also set out to explore the possibilities with regards to realistic simulation in real-time, and potential options for optimization of systems to accommodate this. Another line of investigation that this project necessitated was into what 'realistic' crowd behaviors actually are, and whether or not these behaviors could be simplified down and recreated programmatically without losing the appearance of being realistic.

5 Methodologies

5.1 Research

The key research questions of this project were :

- What is 'realistic group behavior' and of these behaviors have been successfully recreated digitally?
- What are the common requirements for crowd behaviors systems within RTS games?
- What methods of collision avoidance currently exist, and how suitable would they be for the system this project aimed to produce?
- What methods of pathfinding currently exist, and how suitable would the be for the system this project aims to produce ?

To answer these questions, research was carried out using a number of different sources, such as research papers, video games and existing projects and codebases. The main findings from this research were :

- Many of the behaviors observed in crowd situations can be represented mathematically, in many cases this representation is relatively simple.
- Even if isolated, these behaviors can appear visually complex, and imply some sort of intelligence.
- In order to be suitable for use within an RTS game, it would be desirable for a system to be capable of simulating about 400 agents.
- A number of existing collision avoidance systems have been conceived, each with their own positive and negative aspects with regards to the project.

- While there are a number of potential existing pathfinding systems, one in particular, *Dijkstra's Algorithm* may be the most suitable for this system.

5.2 Implementation

Development of the system was a largely iterative process, adding a small component, refining its implementation and then moving on to the next component. This iteration was interspersed with research in order discover possible methods for implementation, or alternatives to planned components when initial ideas were found to not be suitable. This approach was chosen to improve the chance that at least one component could be implemented and presentable at the end of development, rather than many half-finished components that do nothing to present work done. For the most part, this approach was successful, allowing development to fully focus on each component as they were added and meant that any issues faced at any stage could be given full attention, and were only relevant to the current stage of the iteration, saving the time of having to find the potential source of an issue. The aspect of leaving some research of potential options until ends of unsuccessful iterations was also beneficial as experience and finding from work up until that point could guide research, providing more practical results. Despite the positives however, hindsight reveals a number of factors that were problematic with this approach. Firstly, the separation of implementation of components often lead to problems where previous components were not fully designed with future components in mind, meaning time was often lost reengineering work that had already been completed. This problem could be overcome by better structuring the process, planning the order of implementation as well as taking the time to fully consider all of the shared aspects of each possible component. A second issue arose because the process aimed to complete each component to completion before moving on, which became problematic when said implementation became difficult. From experience it was found that it could be hard to decide if and when to admit defeat and attempt another option, and the desire to have something working at the end may have led to some options to be abandoned too soon.

The methodology made solving problems fairly simple, as they were most often isolated, and could be faced with a good degree of freedom, as if any attempt to solve it was not successful, a previous, working iteration could be returned to. Most of the problems faced with this project were technical, and the approach to solving them, was most often trial and error in the short term, trying variations of the existing implementation, and failing this alternative options were explored. Both of these practices would draw from initial research, or exhausting those, require further research. This further research could be a lot more focused than the prior research, as a more specific question was being asked, and lessons learnt while trying to solve the issue could help to build more useful questions.

6 Results

6.1 Overview

The original aim of this work was the production of a crowd simulation system, primarily for use in RTS games. The output fulfils this aim, however it is somewhat scaled down, and does not realise the full scope of the initial vision. As well as a technical system, over the course of development, the research questions have been answered, and while not all answers have been directly utilised, they have all played a part in the development cycle.

The overall final system combines a pathfinding system and a local collision avoidance system, which produce movement behaviors for agents. Many of these agents navigating in a shared environment then creates a crowd simulation. The final system however, does not contain a number of desired features, such as 'individuality' of agents, or wider functionality and usability which would make the system a useful tool for developers.

It should also be noted that while in some circumstances the system is capable of running in real-time, larger environments and some other factors limit its viability in this sense.

6.2 Pathfinding

The pathfinding component of the final system utililises *Dijkstras Algorithm* to generate paths for agents among a grid of nodes. *Dijkstras Algorithm* was chosen as it will always give the shortest route between nodes assuming edges in the graph (the cost from one node to another) are not negative, making it reliable, but also because the algorithm is the basis for other pathfinding algorithms such as *A**, providing options for variations in the pathfinding system, which is useful for trying to better optimise the system for real time use.

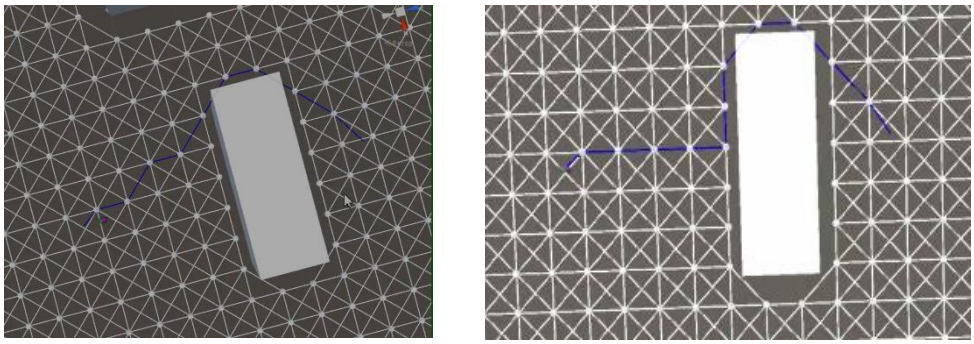


Fig. 1. Examples of the paths built by the system, seen in blue. Left shows path built using Dijkstra's Algorithm, Right shows path where the system prioritises searching nodes that are closer to the goal.

The current implementation of the pathfinding system consistently find the shortest route and can be used for finding routes between nodes on a pre-defined node grid as well as between nodes added during runtime, which enables RTS style, user defined movement. However, it often compromised the real time capability of the system, especially when producing long paths. This however can be improved somewhat by providing it with heuristics (making it operate more like an A* system).

As the paths returned by the pathfinding component are based upon a node system, they are sharp and jagged, and would not produce particularly realistic behaviors should an agent follow then directly. For this reason, another system has been implemented to attempt to smooth these paths out. This system draws inspiration from one of the systems *Left 4 Dead's AI* use, and looks ahead to the furthest visible node, and uses this as a target, bypassing any redundant nodes in between. This makes paths more direct, and while they are still somewhat robotic, combined with the collision avoidance system, smoother, more human behaviors are produced.

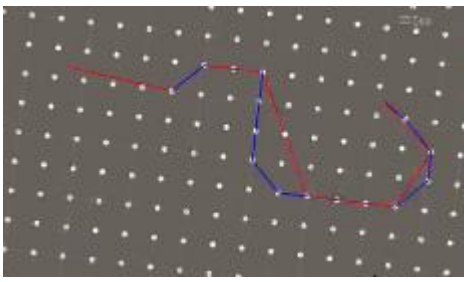


Fig. 2. An example of the difference between the path produced by the pathfinding system (blue lines) and the 'smoothed' path (red lines).

6.3 Collision Avoidance

The collision avoidance system used in the final iteration is also inspired by *Left 4 Dead's AI* system, and makes use of two collision detection areas in front of each agent, one top the left, the other to the right. Whenever something colliders with either of these areas, the agent will begin to turn away from the obstacle, and then return to facing in its desired direction once the obstacle is no longer in the way. This is a simple system that is highly effective at avoiding collisions, and produces smooth movement. The behavior produced by this system is slightly realistic due to the removal of collisions and some of the behaviors produced as a result of this however the local rather than global nature of the collision avoidance, and a lack of 'forward planning' leaves the results lacking in realism when observed closely.

Fig. 3.1. Representation of the arrangement of collision avoidance areas (red and blue rectangles), on each agent (green circle)

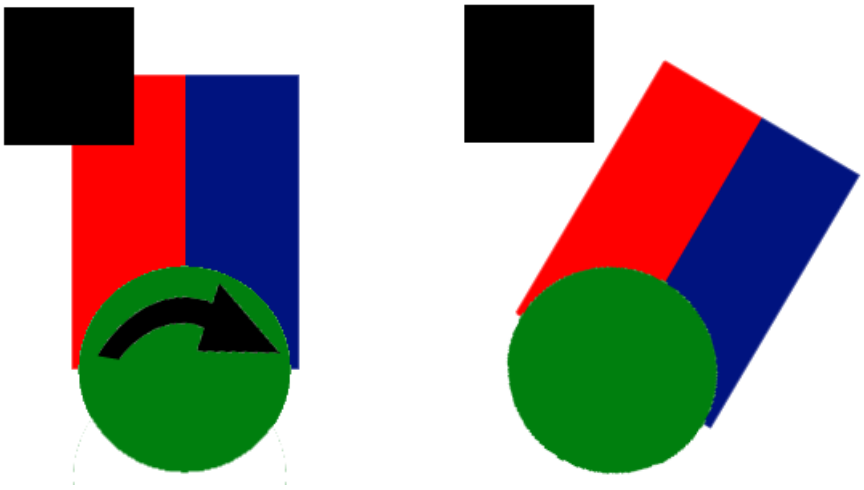


Fig. 3.2. Representation of how collision avoidance systems operates. For example, if an obstacle (black box) collides with the left collision area (red area), the agent will turn to the right, taking it around the obstacle.

The main reason for a collision avoidance component was to deal with avoiding agent on agent collisions, a common problem in real life crowds, and less so on static obstacle avoidance, as the pathfinding system should handle most of that, and it is successful in this regard, removing instances of agent on agent collisions, without causing agents to veer too far off course. This is particularly visible when avoiding head on collisions, as agents will always prioritise turning one direction over the other, avoiding potential incidents of agents both moving the same direction and get stuck in a pattern that takes them further and further away from their desired path. While this component is not very 'smart', and does not have any foresight, it is effective at producing some smart looking and realistic behaviors, although it is likely that testing it in more scenarios may lead to some undesirable or problematic results, and an alternative method, such as one making use of Velocity Objects, as were researched, would provide a much more stable and reliable solution.

6.4 Failures of The Final System

While many aspects of the final output of the project are successful and of a satisfactory standard, there are a some of issues with the system which compromise its suitability for use. These issues are:

- The collision avoidance system is not reliable, and often leads to agents straying very far from intended paths, and also, in some cases becoming stuck on an obstacle. Successful implementation of a system similar to *Clear Path* (Guy *et al*, 2009) would be expected to overcome these issues.

- The pathfinding system appears to be unable to calculate more than one route simultaneously, and attempts to do so will result in errors and agents being left without paths. This limits the systems suitability for use within RTS games. While potential reasons for the issue have been speculated, a possible solution could not be found.

7 Discussion

7.1 Realisation of Aims

The development of the process has been informative with regards to the question of whether or not what had been set out to be achieved was realistic and achievable. While we already knew that a number of individual systems have been created to simulate different aspects of human locomotive behaviors and interaction of these behaviors in a crowd environment, and that these are often capable of doing so in real time, and that systems for producing similar behaviors within video games also exist, we have come on to find that the combination of these individual systems is in many cases possible, and opens up a number of possibilities for systems of varying degrees of realism and complexity. In this sense, it could be argued that the project aim is realistic, and that the creation of a realistic crowd simulation is not only possible, but approachable in many different ways. However, some of the other aspects of the initial aim have been found to be potentially less achievable. While a number of individual systems have been seen to run in real time in some instances, they do not in others, such as the use of Dijkstra's Algorithm for pathfinding system becoming less and less viable as the set of nodes used increased. Because of this, the hope that the system being produced could be versatile and used for a range of applications should be deemed less realistic.

7.2 Other Findings

The outcome of the project also suggests that a single complex systems may not be needed to produce some of the more complex behaviors, as a combination of simpler systems can sometimes produce similar behaviors, as also suggested by *Reynolds Boids model* (Reynolds, 2001). This opens up a possible investigation into exactly what is

possible using this approach, as while complex systems may produce much more desirable outputs, their complexity may limit accessibility, as was found with our attempts to implement a system using Velocity Objects, where a lack of understanding of the mathematics and theory of the system made implementation ultimately not possible, a combination of more easily understandable components, that could produce relatively similar quality outputs would open up said functionality to more developers.

7.3 Potential Application

With regards to the actual system produced, with some refinement it has potential for use in both personal games projects, and as a tool for other games developers, while also providing a good foundation for other specialized systems, as thanks to the iterative development approach and lessons learnt from this, the current system is modular in nature, providing opportunities to mix and match components to suit the needs of a potential variation. With this functionality, systems could be produced for a number of uses, many of them beneficial to society such as simulations within emergency training systems or architecture and urban planning for example, where the use of virtual simulations saves time, money and can allow for better services as many scenarios can be observed quickly and easily.

8 Conclusion

At the end of this project, it has become apparent that the specific expected outcome, to create a versatile, multiuse crowd simulation tool, of the investigation may have been beyond the scope of what was achievable, simply due to the size and variation of needs of possible uses, but an example of a potential system for just one of these possible uses has been created, although it is lacking in polish and depth. Despite being unable to fulfil the expected outcome, a large amount has been learnt about potential development structures for such a task, what kind of similar technology exists and the possibilities and limitations in such a field.

Using the foundation set by this project, work on making the system more efficient and useable as a shareable tool for other developers would open up enticing further investigation, and see that the investigation carried out so far finds real world applications.

9 References

Reynolds, C [2001] Boids Background and Update. Available from: http://www.red3d.com/cwr/boids/ [Accessed 28 November 2015]

Dijkstra. E [1959] A Note on Two Problems in Connexion with Graphs. Niimerische Mathematik 1, pp. 269-271.

Booth, M [2009] The AI Systems of Left 4 Dead. Available from : http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf [Accessed 13 April 2016]

S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey [2009] ClearPath: Highly Parallel Collision Avoidance for Multi-Agent Simulation. Computer Animation, Conference Proceedings, pp. 177-187.

10 Bibliography

Video Explaining Dijkstras Algorithm
https://www.youtube.com/watch?v=WN3Rb9wVYDY

Project Description

My aim for this project is to create a system and/or pipeline for real time crowd simulation, with a focus on human crowds for use within RTS strategy games and for both passive and user controlled crowds, but aiming to create a system versatile enough to fulfil a wider functionality (e.g. other animals and their behaviours, movement in 3 dimensions such as for birds or spacecraft for example)

The system will produce dynamic and realistic behaviours to add depth/life to games (peripheral crowds, city simulation) and/or to create interesting, dynamic gameplay (players having to predict / control behaviours in an emergency situation simulator, realistic enemy behaviour)

Deliverables:

- A unity 'tech demo' to showcase the functionality of my final system.
- A unity asset pack containing all components for an end user to utilise in their own projects.
- User documentation to help users fully utilise the software
- "Stretch goal" : Repackaging the functionality into a c++ library for "plug and play" usability with other systems.

Research and background

Initial research was focused on existing examples of crowd simulation. I was already familiar with the Boids Model[1] which creates interesting and usually realistic swarm behaviours from 3 basic 'rules', and when optimised can be fairly 'cheap' to run making it good for real time applications. While I aim to produce slightly different behaviours to those of the Boids, the method of building up individual agent movements from a number of individual movements based of each behavioural rule is a method I would at least like to try and use to implement my own system.

During my research I came across a tool called *Miarmy*[2] which creates some very realistic human behaviours, and has a very wide range of functionality, much of which outside of the scope of what I hope to produce, however it is a Maya plug in and has been created for use in animation, and so is less suitable for real-time applications. However some of the systems it utilises to create behaviours may be useful for use in my own project, such as finite state machines to move between behaviours.

While I have also found a number of other systems available, due to them being commercial, and often designed for specialist industries (emergency services in many cases) it is difficult to get much more information about the functionality and specifics of these systems, however it is safe to assume that while these systems produce reliable and realistic output (due to the fact they are used by emergency services etc. where these factors, may, in cases make the difference between life and death if

sed for training for example), they are also highly bespoke, and "black box" systems, and so do not allow much re-usability or adaptability for users.

Besides existing full systems, I also looked into the range of possible methods and "sub systems" that I could adapt and combine to create my system. A number of different solutions have already been produced for a number of different problems, one of these core problems is collision avoidance. Simple collision avoidance systems are evident in the Boids model, where a force/acceleration is applied to each model 'away' from any other close agents, however there are a range of other methods. One piece of research[3] looks into modelling collision avoidance by finding the nearest velocity closest to the desired velocity that would not lead to a collision with any nearby agents, after taking into account these other agents would also move to avoid. This system produces desirable effects, runs efficiently even for fairly large crowds, and can be combined with other systems. Another piece of research[4] looks into making collision avoidance "smoother" and more realistic, by damping the speed at which agents move at leading up to collisions, moving faster if there is a big space to move into, slower as they approach a collision, creating 'waves' in crowds of people moving in the same direction. This method only effects 'following' behaviour, but showed me how a realistic behaviour is made up of lots of smaller parts, that are combined to produce the overall effect, like the Boids model, but that each of these individual behaviours can be broken up again. It also showed, as many other examples I looked at, that relatively simple equations can produce complex behaviours.

Finally, there are a number of other sources I have looked at, simply for a better understanding of the task I have chosen to complete, and to better guide my progress with the project. I am yet to fully evaluate these sources, but I have collected them for future reference. These include a paper[5] which, in its introduction discussed the levels of detail expected/required from the sort of system I plan to produce for different end uses, something for me to consider, especially when I am keeping the real-time capability of my system, and also started me planning how I can implement dynamic levels of detail (LoD) to achieve this. I have also began looking into the social force model, animal behaviour models as the behaviours produced by these are often already suitable for human simulation, or can be easily tweaked to be so, and this blog[6] dedicated to crowd simulation for a point of reference for crowd simulation in general.

Objectives

By the end of this project I hope to:

- Achieve the creation of a robust, versatile crowd simulation software for own use and for a portfolio.
- Improve my programming and software development practices.
- Develop my problem solving abilities.
- Gain experience in producing professional documentation and other piece of support work for my projects.

In order to complete the project I will need to find out about:

- The methods of existing simulations.
- Equations and other representations that can be used to create realistic human behaviours, as well as what is considered human behaviour in order to compare my results.
- Options in terms of "streamlining" my system, in order to make it as suitable for real-time applications and use in a wider system as possible.
- The importance of and use of crowd simulation in games, such as what "players" respond to most in terms of realism, levels of detail, interaction etc.

I personally intend to learn / gain:

- About general system programming, to improve my programming ability, especially for programming for larger projects involving many systems working together.
- About the possibilities and opportunities in terms of adapting existing systems to be better suited to use in video games.
- What it takes to create realistic simulations and whether it is possible to reduce computational costs of these simulations without reducing this realism
- A greater understanding of the needs and limitations of systems like the one I aim to produce, as well as of systems for video games in general.
- A greater understanding of progress in the 'world' of simulation and create something that utilises or helps towards this.

Methods and Tools

To complete my project I will utilise skills gained from my course this far, my own understanding and research, and use available resources to gain new skills I may need.

So far on my course I have gained experience using the Unity Games engine, which I plan to use as an environment to complete a majority of my project. My experience thus far has shown me the capabilities and limitations of the engine, one of the main capabilities being it's handling of 3d visuals and UI's, allowing me to focus on the 'under the hood' development of the system. Through my course I have also gained experience of both c++ and c# in different environments and this has given me a good understanding of coding practices and this should assist me in producing versatile, clean and functional code. The experience of c++ is also the main reason I have set my stretch goal to produce a c++ library from my output from unity, as I feel I have enough understanding to effectively port the code.

Risks and Issues

Risk	Mitigation	Contingency
Unity's limitations may hinder progress.		Begin porting toc++earlier, and work within the DirectX framework to give more flexibility
Aspects of my project may require knowledge outside of my current scope.	Focused research throughout the project should provide me with specific knowledge, or at least relevant knowledge that allows me to continue work without a complete understanding of a subject.	

Required Specialist Resources

Input from those with a knowledge of crowd behaviours would be useful but not vital, to give feedback on the output of my project as it progresses to help me produce more realistic results.

If progress goes well, and the core system is finished with time to spare, it would be nice to begin adding animations / models to create a more polished system and better test LoD, which it would be nice to have someone with specific skills produce.

Monthly Project Plan

October	Proposal submitted by 15/10/2015 Implement basic agent behaviour Begin implementing agent Interaction	8 days 7 days 7 days
November	Polish agent interaction and test behaviours with varying group sizes Begin work on pathfinding Combine pathfinding with existing behaviour Polish progress so far	3 days 7 days 7 days 7 days
December	Work on research report and extra time to complete previous objectives that may be unfinished Research Report hand in 10/12/15 Begin work on agent individuality (adding variables) Utilise agent variables for behaviours	10 days 7 days 10 days
January	Begin work on UI and polished debug system Catch up for previous objectives, and prepare some scenarios for the demo	7 days Rest of Jan
February	Prototype demo in class 03/02/16 or – 05/02/16 Begin work on LoD systems Work on feedback from demo	7 days 7 – 14 days
March	If at a suitable stage, begin work on porting to c++ library	Unsure
April	Hand-in 14/04/2016	

[1] Reynolds, C Boids Background and Update http://www.red3d.com/cwr/boids/ [Last updated 6 Sept 2001]

[2] More information on Miarmy at http://www.basefount.com/miarmy.html

[3] Guy, S et al [2009] ClearPath: Highly Parallel Collision Avoidance for Multi-Agent Simulation available at http://pcl.intel-research.net/publications/clearpath_sca2009.pdf

[4] Lemercier, C et al Realistic following behaviors for crowd simulation available at https://spiral.imperial.ac.uk/bitstream/10044/1/21595/2/EG2012.pdf

[5] Badler, N et al Real Time Virtual Humans (Abstract) Available from http://www.cis.upenn.edu/~badler/bcs/Paper.htm

[6] http://crowdsimulation.blogspot.co.uk/

Introduction

Having considered the aims of the project, a number of research questions have been set, the findings of which are documented in this report. These questions are:

- What is 'realistic group behaviour' and which of these behaviours have been successfully recreated digitally?

- What are the common requirements for crowd behaviour systems within RTS games? (e.g. How many agents should the system be able to handle)

- What methods of collision avoidance currently exist, and how suitable are they for my system?

- What methods of pathfinding currently exist, and how suitable are they for my system?

Research Methods

To answer these questions both primary and secondary research have been carried out. The primary source consisting of gathering data from time spent playing relevant games and testing implementation within Unity. The first source, game analysis was chosen as it gives data directly linking to the project aims, and can help guide and inspire implementation, while testing implementation ensured the findings of research could realistically be used. Secondary sources included reading research papers and analysing data from other sources, and this was useful for finding existing forms of implementation, and feedback regarding its results without having to do it personally, as well generally expanding understanding of the subject area of the project.

What is Realistic Group Behaviour?

In order to create a simulation that produces realistic results, it was first decided that research into observations on the behaviours of real life crowds, and some of the phenomena that occur in crowd situations, was needed in order to find and understand these behaviours to allow for robust implementation and recreation, as well as to set criteria against which to compare and judge output. While researching this, many findings came from papers and work into simulation of these behaviours, and hence ways of recreating these behaviours digitally. From these sources it was found that in most cases behaviours can be represented mathematically, so as this project is computing and technology focused, rather than sociologically or physiological, This research disregards the reasons and explanations for these behaviours in real life, in favour of simply noting the visually observable properties, and the mathematic descriptions of these..

One of the first behaviours discovered via the research was the interactions and 'flow' of people at crossing points that create high congestion. Many of the dynamic collision avoidance systems researched would demonstrate their output in these instances with either 4 crossing lanes of people, or with a circle of agents all aiming to move to the point directly opposite them on the circle. Most of the simulations researched created similar effects (see Figures 1, 2) and these effects mirror real life (see Figure 3) fairly closely.

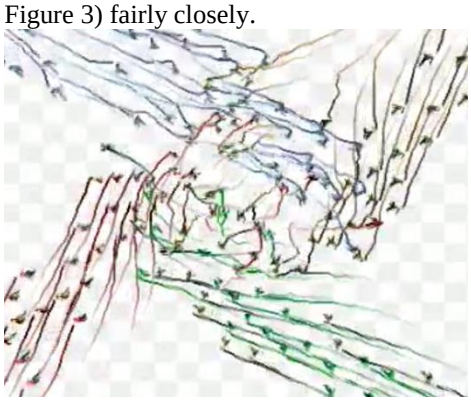


Figure 1: Screenshot from https://www.youtube.com/watch?v=lGOvYyJ6r1c showing continuum crowds simulation

Figure 2: Screenshot from https://www.youtube.com/watch?v=Hc6kng5A8lQ showing *ClearPath* collision avoidance

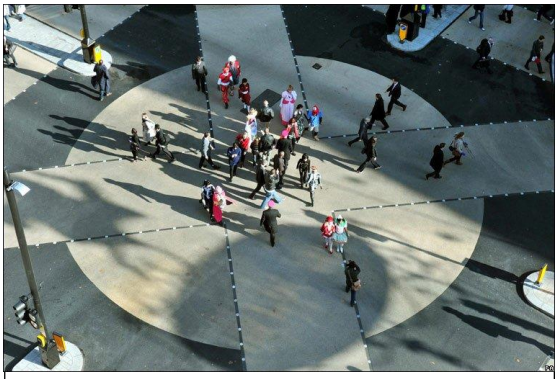


Figure 3: 'X-Crossing' at Oxford Circus. For Video visit http://news.bbc.co.uk/1/hi/england/london/8337341.stm

To create this behaviour realistically, agents need to be able to plan ahead somewhat, and find smooth paths. However although the two examples shown in Figures 1 and 2 aim to keep agents moving in order for smooth motion, in the video referenced in Figure 3, it can be seen that in real life people may actually stop and wait.

Another interesting behaviour discovered through the research is that of 'following', something that if implemented correctly has the potential to both increase the realism of the simulation and make it much more visually appealing. It was also found, through test implementations of a basic system that for simply creating a functioning simulation, there will need to be consideration of at least some form of implementation of this kind of behaviour. This is due to the fact that in the test, agents did not handle collision avoidance well if moving in the same direction as the agent they were trying to avoid. One paper[1] documents some experimentation into the phenomena exhibited in real life, and then simplifies this behaviour down to an equation, which produces very realistic results digitally, this behaviour being the forming of 'waves' within groups of people, as they slow down or speed up depending on the space that opens up in front of them. The fact that this behaviour has been simplified down to a single equation makes it appear to be suitable to be easily combined with any other behaviours the system might be trying to reproduce.

A number of the behaviours researched are those that relate to collision avoidance, one of the two aspects deemed most important to have suitably implemented into the simulation, the other being pathfinding. While creating realistic pathfinding would be a bonus for the system, I believe that with realistic 'local' behaviours for agents, such as those discussed in this part of the report, even a fairly simple pathfinding system will produce suitable results, and this is why this section has not focused on 'real life' path-finding behaviours and phenomena.

With regard to existing RTS games, what are my target specifications?

In order to guide the project, it was decided it would be beneficial to analyse some existing RTS (Real Time Strategy) Games, to find out some of the common require-ments and capabilities of these, to set some specifications. The main specification to set was how many agents my simulation should be able to handle, however through analysis, a few more aspects to consider were discovered. To answer this question three RTS games were analysed, these being *Command & Conquer Red* Alert 2[2] (RA2), Starcraft 2[3] (SC2) and *Company of Heroes*[4] (CoH). However only RA2 and CoH are discussed in depth here, as RA2 and SC2 returned similar results. Firstly, I tested the maximum number of agents possible in each of these games. To do this, some custom games were played, with the aim of simply hitting the unit count limit for one player, and extrapolating the results here to take into account multiple players. In RA2, there does not appear to be a 'limit' on the amount of units a player can produce, the only limit would be how many units can fit on the actual level. However it was found that when selecting a large number of units and issuing a move order, not all of the units would move (only about 130 would), leading me to believe that the system for unit movement behaviours only handles this many units, although considering there are a

maximum of 8 possible players in a game, it would make sense to think that the system is actually able to handle closer to 1000 units movement. When considering this number, it should be considered what this system actually does. 'Idle' units are totally idle, they do not react to other units in most cases, unless it is to attack units that come close, but even then the unit will not move to do this. Moving units take the quickest route to their destination and while they sometimes appear to move around other units in their crowd, this is not very realistic and appears to simply assume other moving units are simply stationary objects each 'tick'. These facts combined make the system used in this game sound fairly simplistic, but considering this game was released in 2000, I feel it is suitable to use this game as a benchmark for the more complicated system I wish to produce due to the progress in the ability of computers since then. As noted earlier, SC2 is similar to RA2 in terms of capability, with idle units essentially totally idle, beside a few exceptions. The only notable difference is that there is a unit limit 200 'supply' (the games measure of army strength), with a maximum number of units per player being 400 (one particular unit is worth 0.5 supply). At the opposite end of the spectrum, in terms of what the unit movement system is capable of, is CoH, a game whose system produces results similar to those I envision for my system. Firstly it was found that, in an 8 player game, the maximum number of units possible would be around 350, assuming all players only used infantry. Comparing this to SC2's unit limit suggests that aiming to produce a system that can handle between 300 and 500 agents would be sensible. On top of this, taking a look at the capabilities of the CoH system has shown some features that would be desirable for a system, especially one that would be used for a military RTS. Some of the features in CoH include units fleeing when losing, units moving autonomously to better positions (e.g. behind cover or for better firing positon) and behaving differently while in different states, for example crawling and moving slowly while 'pinned' (under heavy fire). These kind of features add a lot of realism to the game and so would be desirable to try to integrate into this project.

What methods for collision avoidance already exist?

As stated in the abstract, collision avoidance and pathfinding are the core aspects of the system this project aims to produce. It was quickly realised that creating methods for handling these two areas from the ground up would not be realistic for the timeframe, and that there were likely a wide range of existing methods that could be utilised and modified to suit the needs of the system. This section covers a number of these methods.

The first method researched was the *Boids*[5] system as a potential method. This method was chosen first as I have experience of implementing it in a previous project, and found it created some interesting behaviours. As an overview, the *Boids* model creates realistic swarm behaviours, by simply combining velocities which are 'created' as a result of 3 basic rules. These basic rules are 'Separation', 'Alignment' and 'Cohesion'. In terms of collision avoidance, only the separation rule here is of interest. The rule for separation simply finds a vector pointing away from an agent that is too close,

for each agent that is too close and finds the average of all these vectors to give a direction to move which is away from close neighbours. This method is very simple, although like any system that relies on checking against all other agents it is $O(n^2)$ (quadratically complex), unless the amount of agents each agent checks for is limited in some way. The simplicity of this method makes it appealing for use due to the real time limitation on the project, however although it produces realistic swarm behaviours, these do not translate very well to human behaviour and in this system, agents don't plan ahead in their movements, like humans do. It is worth noting however that the modular approach of the *Boids* system, is one which will likely guide development, in which the final 'output' is produced by combining the output of each of the separate behaviours of the system, if this is possible.

Knowing that the *Boids* model would likely not provide the suitable output, alternate methods were pursued. Searching for 'real time multi agent collision avoidance' returned a video[6] demonstrating a system called ClearPath, which makes use of 'Velocity Objects' (VOs). The essential idea of VOs is that if an agents current velocity lies within

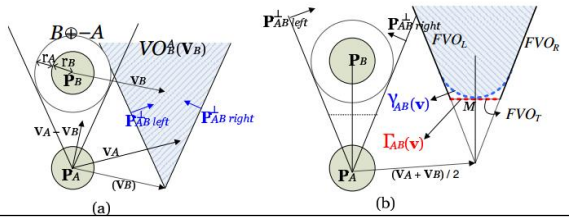


Figure 4: Diagrams detailing Velocity Objects from ClearPath Paper[7]

a VO shape, then there will be a collision, and to avoid this collision the agent should find a new velocity, which lies on the edge of the shape. The VO shapes are, in their most basic form cones, with the apex being the position of the current agent (see figure 4). The behaviours exhibited in the video are much more like what is envisioned for the project, and the information about the Frames Per Second (fps) at high numbers of agents suggest that this system can be viable for real time use. For these reasons it was chosen to settle on trying to make use of velocity objects in my own system, and to make sure this would be viable, testing in Unity was carried out. I understood the theory of the system but it took some time to understand the maths, specifically in producing the VOs. For these reasons the test implementation was inefficient, incomplete and unable to accurately recreate the system. This lead to the search for existing implementation to utilise, which led to finding a library[8] that handles RVO's (Reciprocal Velocity Objects), which will help streamline implementation into the project.

An alternative collision avoidance method that was found is that described within a social forces model[9], which as a whole is similar to the *Boids* model, combining a number of different desired behaviours to guide an agent. The collision avoidance behaviours in this instance is based around agents becoming 'uncomfortable' as they come too close (defined by a 'private sphere') to 'strangers', and so other agents apply repulsive effects. Within this model, there is also a behaviour that describes how agents will

avoid obstacles (such as buildings, street edges etc.) where the borders of these obstacles will also apply repulsive effects.

What methods for pathfinding currently exist?

After collision avoidance methods had been explored, ways of handling path or route finding for the agents within my system was looked into. It should be noted here that although at an high level, collision avoidance and pathfinding may be very similar concepts, in the case of this paper, collision avoidance is considered to be the method with which agents alter their desired routes to avoid local collisions with, in most cases dynamic, obstacles, whereas pathfinding is considered to be the higher level system that sets an agents previously mentioned desired routes, between their current positions and either predefined or dynamically set target positions.

Before researching the decision was made to focus on node based methods, and as was stated earlier, the desired method is simple and focused on being fast and real time suitable, as the other behaviours should provide the interesting behaviours. Found paths should also be the quickest as well as being reliably predictable in order to work within a gameplay setting (a player would likely become frustrated if units they control do not move as they expect). It has also been decided that the nodes for navigation will be manually placed by a designer rather than generated by the system due to project scope, and also a means to allow a designer to have more control over the output of the system dependant on the needs of the project it could be used for.

After a search for pathfinding methods, a common result was *Dijkstra*'s *Algorithm*[10], a method for finding the shortest path between nodes within a graph. For the algorithm to be run, it simply needs to be supplied with the possible paths between nodes and the weights of these paths (could simply be equal to the distance or calculated to consider a number of variables). The algorithm starts by setting the cost of all routes to be infinity, and then finds the cost of each possible path from the start node, and sets these costs as the new cost of the route (as they will have a cost less than infinity), and repeats this for this step for the cheapest, not yet checked node, each time replacing a previous cost if it finds a cheaper alternative. The original algorithm runs in time $O(x^2)$ (where x is number of nodes), as it finds the shortest route from a start node to all others. This could cause problems for scenarios where node graphs are large and/or complex, and may affect the real time capability of the method, however, modifying the algorithm to stop when finding the quickest route to a particular node, and for instances where the graph isn't going to change, once quickest routes have been found, they can simply be stored and re-used. Keeping in mind the RTS application however, where a user will set destinations on the fly, this will not be a useable technique.

Due to the potential issues with using *Dijkstras Algorithm*, another potential method was found. The next suitable method found was *A**[11], an extension of *Dijkstras Algorithm*, developed to make use of heuristics to improve performance. The cost algorithm for the method:

$$f(n) = g(n) + h(n)$$

where $g(n)$ is the cost of getting from the initial node to a node n, and $h(n)$, is the cost of getting from n to the goal. These heuristic functions can vary dependant on the problem, and so this could provide a good customisation option to users of the system,

but this can also be as simple as the distance to the goal for *h(n)*, so would not require the user to have an in depth understanding of suitable heuristics. The use of these heuristics helps the algorithm find a route in less cycles than *Dijkstra's* original (see figures 5 and 6), and so would likely be more suitable for real-time application.

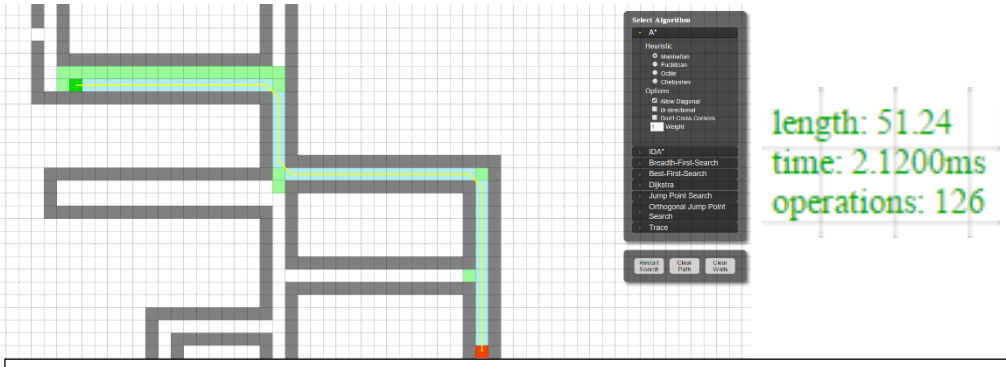


Figure 5: Visualisation of A* pathfinding method (left) and Information about output (right). (Operations shows number of 'steps' taken to find route)

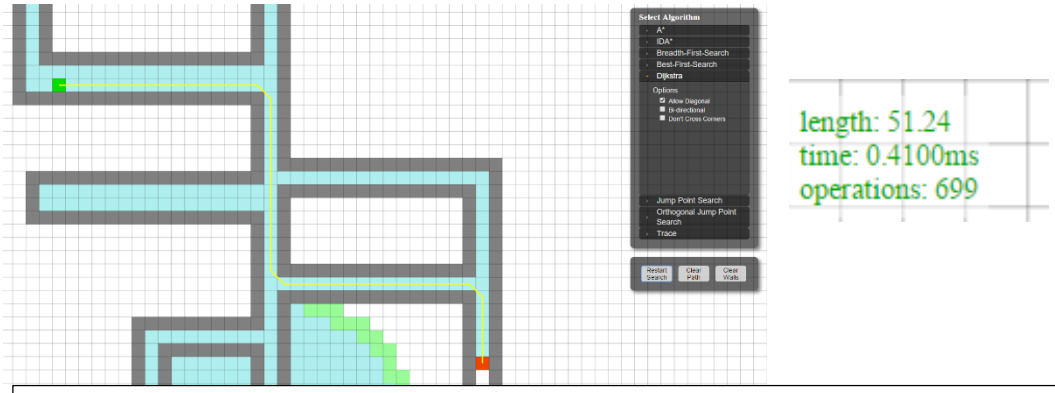


Figure 6: Visualisation of Dijkstra's Algorithm pathfinding method (left) and Information about output (right). (Operations shows number of 'steps' taken to find route) (using tool at http://qiao.github.io/PathFinding.js/visual/)

Conclusion

In light of the research conducted, the focus of the project has changed slightly. Initial aims were to develop a bespoke system, for a fairly vague need, however now the aim is to work towards combining existing individual systems to fulfil a more specific goal, which has been guided by the research into existing RTS games. After struggling to work on a unique implementation of the Velocity Object system and finding the libraries available, it is likely that new project aim will revolve around modifying existing 'components' to work efficiently alongside other, and this should open up the time to tailor the system to be a more usable end project, offering customisation options to a user. Initially the project will aim to simply combine the discussed Velocity Object system and an A* pathfinding system. From here, if the output of this combination does not produce desired results, the addition of some of the social forces model behaviours would be an option in order to improve the simulation, especially as these are already known to combine effectively, otherwise if results at this point are suitable, work should begin on setting up the system to handle different types of agents, as would be the case in an RTS game, as well as setting up a front end system to allow users to set up levels with pathfinding nodes, spawn points for agents etc.
On top of these aims, based on the analysis of existing games, the system should, at the minimum, be able to handle 400 separate agents in real time, however it would be desirable to have it handle more, and, as the system of CoH produces much more realistic results than the other analysed games, it is an aim to include at least one of the features defined (e.g. fleeing due to stimuli or a range of states that affect behaviour)

References

[1] S. Lemercier, A. Jelic, R. Kulpa, J. Hua, J. Fehrenbach, P. Degond, C. Appert-Rolland, S.Donimian, J. Pettré [2012] Realistic Following Behaviours for Crowd Simulation. Available from: https://spiral.imperial.ac.uk/bitstream/10044/1/21595/2/EG2012.pdf
[2] Westwood Pacific [2000] Command & Conquer: Red Alert 2[computer program]. Published by EA Games [Accessed November 2015].
[3] Blizzard Entertainment [2010] *Starcraft 2 Wings of Liberty* [computer program]. Published by Blizzard Entertainment [Accessed November 2015].
[4] Relic Entertainment [2006] *Company of Heroes* [computer program]. Published by THQ [Accessed November 2015].
[5]C. Reynolds [2001] Boids Background and Update
Available from: http://www.red3d.com/cwr/boids/ [Last Accessed 28 November 2015]

[6] GAMMA UNC. [2009] ClearPath: Highly Parallel Collision Avoidance for Multi agent Simulation. *Youtube* [video]. 01 November. Available from: https://www.youtube.com/watch?v=Hc6kng5A8lQ [Last Accessed 28/11/2015].
[7] S. Guy, J. Chhugani, C. Kim, N Satish, M.Lin , D. Manocha, P. Dubey [2009] ClearPath: Highly Parallel Collision Avoidance for Multi-Agent Simulation. Available from: http://pcl.intel-research.net/publications/clearpath_sca2009.pdf
[8] J. Berg, S. Guy, J. Snape, M. Lin, and D. Manocha. [2015] **RVO2 Library:** Reciprocal Collision Avoidance for Real-Time Multi-Agent Simulation

Available from: http://gamma.cs.unc.edu/RVO2/downloads/

[9] D. Helbing, P. Molnár [1995] Social force model for pedestrian dynamics. Available from: http://arxiv.org/pdf/cond-mat/9805244.pdf
[10] E. Dijkstra [1959] A Note on Two Problems in Connexion with Graphs. Available from: http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf
[11] P. Hart, N. Nilsson, B. Raphael. [1968] A Formal Basis for the Heuristic Determination of Minimum Cost Paths. Available from: http://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf