

Game AI R&D Project

AI Research

Synopsis: This document outlines research on various game AI techniques and concepts, for use in the design and implementation of a goal-based AI system.

Reference: GameAI.Research v1.0

Date: 7 April 2006

Author: Gavin Bunney & Tom Romano

Status: Definitive



immersive ai engine

Document Control

Version History

Every change to this document is logged in the table below.

Ver.	Date	Author	Description
v0.1	2006-03-26	Gavin Bunney	Initial document creation
v0.2	2006-04-04	Gavin Bunney	Altered layout and added general AI concepts
v0.3	2006-04-04	Gavin Bunney	Added goal-based techniques information
v0.4	2006-04-04	Tom Romano	Added limitations and best practise information
v0.5	2006-04-06	Gavin Bunney	Added decisions under uncertainty
v0.6	2006-04-06	Tom Romano	Added initial A* path finding information
v0.7	2006-04-07	Tom Romano	Changes on formatting / glossary / references
v0.8	2006-04-07	Gavin Bunney	Added project abstract / introduction / synopsis
v0.9	2006-04-07	Gavin Bunney Tom Romano	Changes based on internal review feedback
v1.0	2006-04-07	Gavin Bunney Tom Romano	Definitive Issue

Project Abstract

Implemented game AI, particularly in RPG/MMORPG games, is very script based. The NPC's walk in a set path, speak with set scripts; the mobs have scripted actions. The concept behind the project is to create a more realistic AI, to both provide unpredictability in NPC behaviours, and to immerse a player in the game world.

The project is to research AI techniques, design and implement a goal-based AI system. Goal-based AI is a technique used to create NPC's which act as real players; in that they are given an objective to achieve - e.g. Given a goal of "rake leaves" it is up to the NPC to work out how to achieve their goal, whether through buying a rake, stealing one, or killing another NPC for their rake.

The implemented AI system will be in the form of various classes, created in C++ and Torque Script for the Torque Game Engine, version 1.4. For more information on the Torque Game Engine, visit <http://www.garagegames.com>.



Contents

1	INTRODUCTION	6
1.1	PURPOSE	6
1.2	SCOPE	6
2	AI CONCEPTS	7
2.1	DEFINING AI	7
2.2	STRONG AND WEAK AI PRINCIPLES	7
2.3	BEHAVIOUR	7
2.3.1	Perception	7
2.3.2	Action	8
2.3.3	Reaction	8
2.3.4	Learning	8
2.4	SPECIFIED & UNSPECIFIED BEHAVIOUR	8
2.4.1	Deterministic AI	8
2.4.2	Nondeterministic AI	8
3	LIMITATIONS.....	9
3.1	INDUSTRY BEST PRACTISE	9
3.1.1	Cheating.....	9
3.1.2	Finite State Machines.....	10
3.1.3	Fuzzy Logic.....	10
3.1.4	Bayesian Networks.....	10
3.1.5	Path Finding Techniques	11
3.1.6	Other Techniques.....	11
4	GOAL-BASED TECHNIQUES.....	12
4.1	GOAL-DIRECTED REASONING.....	12
4.2	DECISION-MAKING PROCESS	12
4.2.1	Game Analysis	12
4.2.2	Goal Formation, Evaluation and Prioritisation	12
4.2.3	Plan Formation and Evaluation	12
4.3	EXAMPLE GOAL-BASED AI IN GAMES	13
4.3.1	No One Lives Forever 2	13
4.3.2	Elder Scrolls 4: Oblivion	13
5	DESCISIONS UNDER UNCERTAINTY	14
5.1	BAYES' THEOREM	14
5.2	BAYESIAN NETWORK.....	15
5.2.1	Structure	15
5.2.2	Inference	16
5.2.3	Implementation	17
5.2.4	Limitations.....	17
6	A* PATH FINDING.....	18
6.1	DESCRIPTION.....	18
6.2	IMPLEMENTATION DETAILS.....	18
6.2.1	Overview	18
6.2.2	Properties of Each Node	18
6.2.3	Goal	18

6.2.4	Heuristic.....	19
6.2.5	Fitness	19
6.2.6	Other A* Components	19
6.3	PSEUDO CODE.....	19
6.4	WEAKNESSES OR LIMITATIONS.....	20
7	ACRONYM GLOSSARY	21
8	REFERENCES	22

List of Figures

Figure 5-1: Basic Structure	15
Figure 5-2: Simple Networks.....	16

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to outline various game AI techniques and concepts for use in the design and development phases of the Game AI R&D Project: immersive AI engine. This document will form the basis for various design justifications as required, being a summary of various research topics.

1.2 Scope

The scope of this document is limited to various game AI techniques which are seen to be applicable during the design and implementation of the immersive AI engine. They are focused around the concepts of Seek (A* path finding), Attack (Bayesian Networks) and Interact (Decisions Under Uncertainty).

2 AI CONCEPTS

2.1 Defining AI

With advancements in both hardware and software technologies, visual immersive game environments are now commonplace. The focus for game developers now is shifting towards creating computer characters which act, react and interact with the game world in a seemingly intelligent manner.

According The American Heritage Dictionary of the English Language, Fourth Edition (2000), artificial intelligence is "The ability of a computer or other machine to perform those activities that are normally thought to require intelligence". This leads to the difficulty of defining what intelligence is, in order to develop and understanding of how we can portray it within a virtual environment.

Paul Tozour stated in AI Game Programming Wisdom (2002, 10), "What we need is not a generalized 'intelligence', but context-dependent expertise.... Each game is its own unique 'evolutionary context', so to speak, and a game's AI must be evolved within that context." The aim of an AI system should be then, to produce immersive behaviours, based on the context of the characters environment.

2.2 Strong and Weak AI Principles

In a pure form, AI would learn and adapt to any problem it faces, like a human, and react based on its own conscience and emotional state. This is commonly described as Strong AI.

Within a game environment however, it simply isn't feasible to create pure learning characters; creating characters which react to their environment, in a human-like fashion, is achievable with current hardware and software technologies. Game AI is seen as Weak AI; for a further description on limitations faced within game environments, see Section 3, Limitations.

2.3 Behaviour

Kenneth Finney describes in his book Advanced 3D Game Programming All In One (2005, 151), that when considering intelligence, four types of behaviour are useful in games, thus require simulation:

- Perception
- Action
- Reaction
- Learning

2.3.1 Perception

According The American Heritage Dictionary of the English Language, Fourth Edition (2000), perception, from a psychology standpoint is "The neurological processes by which such recognition and interpretation are effected". It is the ability to detect and understand changes in an environment.

Within a game environment, it is desirable for computer characters to acknowledge and perceive changes; such as entity detection, entity destination and sound recognition. An example of bad perception is where a grenade is thrown in front of a computer character, but they simply remain in the current position.

2.3.2 Action

The tasks which a computer character performs by its own choosing are action behaviours. Within a game environment, actions are commonly based on a set of rules and constraints; patrol here, eat this etc.

2.3.3 Reaction

Reaction is the tasks which are executed due to a trigger within the game environment. This is a combination of the perception and action behaviours, in that it forms the basis for a computer character to execute an action on various game stimuli. Complex reaction systems would choose the best action to perform based on the perception, with simple reaction system executing a set task on a trigger.

2.3.4 Learning

According The American Heritage Dictionary of the English Language, Fourth Edition (2000), learning is "The act, process, or experience of gaining knowledge or skill". Within game environments the most desirable learning behaviour would be to record those actions which work, and those which don't to avoid repeated mistakes.

2.4 Specified & Unspecified Behaviour

There are typically two different ways of representing behaviour within games – deterministic and nondeterministic AI. These two concepts represent specified and unspecified behaviour.

2.4.1 Deterministic AI

Deterministic AI is where character behaviour is specified. It is best described in AI for Game Developers, by Bourg and Seemann:

Deterministic behaviour or performance is specified and predictable. There's no uncertainty. An example of deterministic behaviour is a simple chasing algorithm. You can explicitly code a nonplayer character to move toward some target. (2004, p. 2)

The deterministic techniques are efficient, easy to implement, test and debug, as all characteristics are predetermined within the game code. They are however, due to their scripted nature, predictable and it becomes quickly foreseeable what action a character is going to perform next.

2.4.2 Nondeterministic AI

Deterministic AI is where character behaviour is unspecified. This creates a much more immersive environment, but does produce other problems for testing and debugging, as characters may not react the same way through each test phase. As described in AI for Game Developers, by Bourg and Seemann:

Behaviour has a degree of uncertainty and is somewhat unpredictable (the degree of uncertainty depends on the AI method employed and how well that method is understood). An example of nondeterministic behaviour is a nonplayer character learning to adapt to the fighting tactics of a player. (2004, p. 3)

3 LIMITATIONS

Artificial Intelligence in games has always been governed by certain constraints. Modern technology and AI techniques allow us to argue that a computer can be made to “think”. “Pure” AI however, the simulation of human intelligence, many believe can only be achieved through the replication of emotions and feelings; the belief that emotions are integrally tied into what it means to be intelligent. Kenneth Finney describes in *Advanced 3D Game Programming All In One* (2005, 137), that “In order to experience feelings or emotion ... a thinking being needs to be *self-aware*”. Modern technology and current AI techniques have, as yet, been unable to make thinking beings “self-aware” within games. However, current game AI technology has allowed designers to create a perception of intelligence, to create characters which react as if intelligent, but who are still tied to the limitations of their various implementation techniques.

One of the other limitations of AI development within games today is thought to be CPU processing power. With so many CPU cycles required by the graphics and physics engines within present-day games, allowing further processing time for usually CPU cycle intensive AI calculations is just not feasibly possible. Of course, with the next generation gaming platforms such as the Sony Playstation 3, with multiple concurrent CPU process cores, we should hopefully see this limitation expire in the near future.

Yet another limitation development teams recognise is that, creating dynamic AI agents in games increases coding, debugging and testing difficulty. After all, if an AI agent is performing some dynamically changing action, then how can a development team test to ensure that all the outcomes are in fact successful? Of course, not all possibilities can be properly accounted for in some instances, this is especially relevant in terms of game development, which is usually governed by strict budgets and deadlines.

How successful current AI implementations are today, is really impacted upon, by a designer’s definition of Artificial Intelligence. AI in its “pure” form has not yet been achieved because of the constraints of CPU time, increased coding load, budgets and deadlines resulting in agents who do not reflect emotions or “self-awareness”. It is these factors together with others that are the reasoning behind why AI in its “pure” form cannot be fully realised, at this point in time. Yet there are some established techniques or practises within the industry which help to negate these constraints and produce what we call modern game Artificial Intelligence.

3.1 Industry Best Practise

Due to the limitations and the sheer complexity of the problem posed by the creation of Artificial Intelligence, the industry uses a variety of techniques in order to make a computer appear intelligent. Much research has discovered some techniques which can help to create the illusion on intelligence; some of these are outlined below.

3.1.1 Cheating

The one technique that seems to be common to most of these resources is in fact that of cheating. Basically meaning that, developers disclose certain environment information to the computer characters. AI for Game Developers, by Bourg and Seemann provide this example of cheating in an RTS game where the,

“computer team can have access to all information on its human opponents

(location of base, types, number units etc) without having to send out scouts to gather such intelligence the way a human player must." (2004, 3)

The main reason given for this occurrence within modern AI is to ensure that the AI is smart enough to make it challenging for a human player. A human player has the ability to adapt and learn from different strategies, but the computer does not; thus cheating is an effort to re-establish a level playing field. Cheating in games however, if noticeable by the human player, can cause a player to stop playing the game, in saying that so can the level of the challenge the computer is providing. It is a delicate balance but most game players would however, agree that they would still prefer an AI that cheats effectively rather than one that is completely hopeless and not challenging at all.

3.1.2 Finite State Machines

The use of Finite State Machines in computer games apparently dates back to the days of the ghosts in Pacman, who were in fact simple finite state machines. A finite state machine (FSM) according to AI for Game Developers, by Bourg and Seemann, is a "machine that can exist in one of several different and predefined states" (2004, 165). A FSM upon activating certain triggers will change states and move from one state to the next. Each state has a set of predefined actions associated with it and a trigger which will move it to its next state. It is in this way that FSM's can be used to subtly mimic human behaviour within games.

For example, if a non player character (herein denoted NPC) in an RPG is to carry out their daily routine they may have a few various states that they exhibit during a day, such as eating, sleeping, working and cleaning. Each one of these states would have an associated routine that would need to be carried out along with a trigger which would cause the states to change. FSM are therefore without a doubt an old, yet extremely useful technique when simulating intelligence in artificial entities.

3.1.3 Fuzzy Logic

Decision making and control within games can be assisted through the implementation of the mathematical technique known as Fuzzy Logic. In Boolean logic within computing, there are two categories that an item can fall into, either True or False (On or Off). There is no in between measures in Boolean logic, however Fuzzy logic gives us the tools to fill the grey area; it enables decisions and controls to be probability based. Everything in fuzzy logic can be considered to be true, but just at varying levels between 0 and 1. A truth level of 1 is absolutely true and a truth level of 0 is absolutely not true (false).

Fuzzy logic enables us to define the middle-ground and also provides mechanisms for quantifying the degrees of truth within the middle-ground. One such AI example illustrating a game's employment of fuzzy logic is its use by NPC's to evaluate the threat posed by there human opponents. The computer can use figures of strength and distance of a human opponent in order to make a calculation, which will determine whether that NPC fights or flees.

3.1.4 Bayesian Networks

These networks, also known as belief networks, according to Paul Tozour in AI Game Programming Wisdom (2002, 7) involves the use of "probability theory to deal with uncertainty and incomplete knowledge of the world". Game AI uses these networks along with Bayes' theorem in an attempt to help deal with "predicting uncertainty". It

takes known probabilities and tries to assign values to combinations of those probabilities. Thus they can be used for example to predict that if a kick from an opponent is on the left constantly then to block on their next attempt. These networks are explored in more detail in Section 5, Decisions Under Uncertainty.

3.1.5 Path Finding Techniques

Some may not consider path finding as part of Artificial Intelligence within games, however it plays an integral part in ensuring intelligent beings are created. After all if a NPC can't walk around a tree in order to attack their human opponent then what use are they? Thus AI agents being capable to plot a safe course, from start to end point, whilst avoiding obstacles and threats, within games is essential to creating intelligent beings.

One of the most common and oldest techniques of path finding implementations is the A* (A Star) method. It involves searching surrounding areas for the least costly path from start to finish, by examining neighbouring states. The A* algorithm is outlined in further detail in Section 6, A* Path Finding.

3.1.6 Other Techniques

Some other techniques not covered in detail include:

- A-Life or Artificial Life – attempt to exhibit natural behaviours by applying universal properties to AI agents
- Expert Systems – capture human expertise within a given specified field
- Genetic Programming – attempt to imitate evolution process through selection and interbreeding on algorithms
- Multi-agent systems – intelligence arising from interaction between multiple competing or cooperating agents
- Situation Calculus – employs first logic to calculate how an agent should react in a given situation
- Production systems – based upon a database with a set of rules
- Neural Networks – machine learning techniques based upon the brain's architecture

These can be investigated further easily on the internet or in detail in the book Artificial Intelligence: A Modern Approach (1995) by Russell and Norvig.

4 GOAL-BASED TECHNIQUES

A technique for creating immersive computer characters is the use of individual "goals" for a computer controlled character to achieve. In a pure form, they are assigned a list of goals and need to determine which to perform at a point in time and how to achieve the goal.

Goal-based AI is an excellent technique for creating immersive game environments, as game designers can create characters which perceive, act and react in the game environment. It allows for the computer characters to perform their own decision making process, creating an unpredictable, engaging environment.

4.1 Goal-Directed Reasoning

Goal-directed reasoning is a technique where a goal is analysed from end to beginning; it is effectively backwards thinking. Every task which could result in the end state is analysed, then each task to achieve this task is discovered. It is an iterative process until a task discovered is the initial state. This can be implemented for a game AI system to form the basis for issuing commands to computer characters – such as giving a character the goal of eat. The computer character would then need to use a goal-directed reasoning process to calculate the best ways to achieve the goal of eat.

4.2 Decision-Making Process

John O'Brien outlines in *AI Game Programming Wisdom* (2002, 375-377) a technique to take the world state, analyse, form goals and create an action plan. This process is similar to the way humans' process problems, thus can be understood and programmed in an intuitive fashion.

4.2.1 Game Analysis

The first stage of a decision making process is the analysis of the environment. It is the process of collecting information about the world state from the perspective of the computer character. It allows for the computer character to ascertain the various priorities and their availability in the current environment state.

4.2.2 Goal Formation, Evaluation and Prioritisation

Once the game environment has been analysed, the next stage is the evaluation of goals, both already existing in a computer characters list and the formation of new ones if required. The key part of this phase of the decision making process is the prioritisation of the goals – those essential to the function of the character must be executed primarily, with the subsequent goals executed after the high priority ones.

4.2.3 Plan Formation and Evaluation

Once the list of goals has been formulated, with varying priorities, the AI needs to decide what order to execute the tasks to achieve the various goals in. This requires analysis of the difficulties for each task versus the priority of the goal.

4.3 Example Goal-Based AI in Games

Recent games such as No One Lives Forever 2 and Elder Scrolls 4: Oblivion, employ goal-based AI for the computer characters in game.

4.3.1 No One Lives Forever 2

Craig Hubbard described the advantages and disadvantage of utilising a goal-based system in NOLF2 in an interview with Action Vault on September 27, 2002:

We chose the goal-based approach because we wanted a living world, with AIs that behave believably and have some purpose other than killing the player... Each individual AI does not need to be scripted to respond to different situations. Goals take care of responding appropriately to whatever situation an AI (agent) finds himself in. Other benefits of goal-based AI are that it's easier to code, and easier to globally modify behaviour. Each goal is its own module, so each behaviour can be tested individually... Once an AI is supplied with a full palette of goals... the AI may behave in unexpected ways. This unpredictability is great for gameplay, but can cause trouble during development and testing.

(<http://actionvault.ign.com/features/den/nolf2qa01.shtml>)

4.3.2 Elder Scrolls 4: Oblivion

Oblivion, the forth in the Elder Scrolls series of games, employs a new AI system named Radiant AI. Radiant AI is a goal-based system and as described by game maker Bethesda Softworks <http://www.bethsoft.com/games/games_oblivion.html>, how computer characters "make their own choices based on the world around them. They'll decide where to eat or who to talk to and what they'll say. They'll sleep, go to church, and even steal items, all based on their individual characteristics."

This system creates an incredibly immersive gaming world, where the computer characters appear to "live" in the game. Such that characters are performing their daily activities whilst you are travelling around. This allows for the focus of the game world to be shifted away from "all about the player", but rather to the entire world, where the player is simply a part of the greater world.

5 DECISIONS UNDER UNCERTAINTY

Any good game design has factors of uncertainty – having a computer character with intimate knowledge of the entire game environment can lead to mundane, unrealistic game play. The better alternative would be to disclose only certain information to the character and have them make a decision based on this knowledge.

Bayesian networks are a method which employs Bayes' Theorem to calculate conditional probabilities. As stated in *AI for Game Developers* by Bourg and Seemann (2004, 245) "Bayesian networks are graphs that compactly represent the relationship between random variables for a given problem." They are an effective method to perform decision making when certain factors are uncertain.

5.1 Bayes' Theorem

Bayes' Theorem is used to calculate conditional probabilities. The following is an excerpt from an article by Paul Tozour in *AI Game Programming Wisdom* (2002, 346), which clearly states Bayes' Rule with a given example:

The foundation of probabilistic reasoning is Bayes' Theorem, which allows you to reverse the direction of any probabilistic statement. Let's say you want to know that chance that it rained yesterday if you suddenly find out that the lawn is wet. Bayes' Theorem allows us to calculate this probability from its inverse: the probability that your lawn would be wet if it had actually rained the day before. Bayes' Theorem is written as:

$$P(A|B) = P(B|A)P(A) / P(B)$$

$P(A|B)$ translates as "the probability of A given that what I know is B" – in this case, "the probability that it rained yesterday, given that your lawn is wet." The theorem allows us to rephrase this in terms of the probability of B given A (" $P(B|A)$ ") and the independent probabilities of A and B (" $P(A)$ " and " $P(B)$ "). These translate into:

$P(B|A)$ = the probability that the lawn would be wet, if it actually rained yesterday

$P(A)$ = the probability of rain, all other things being equal

$P(B)$ = the probability of your lawn being wet, all other things being equal

5.2 Bayesian Network

The purpose of a Bayesian Network is to represent relationships between variables to aid in decision making processes when factors are uncertain. The Bayesian Network has advantages over techniques such as fuzzy logic in that when new information is added to the Bayesian Network, the probability calculations alter, whereas in those monotonic logic systems they cannot change.

5.2.1 Structure

Bayesian Networks comprise of various nodes representing variables, with links between them indicating the causal relationships. Figure 5-1 below illustrates A having a causal relationship with C; B having a causal relationship with C. This demonstrates that A can cause C, B can cause C or A and B can cause C.

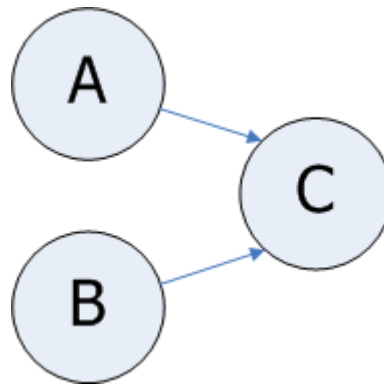


Figure 5-1: Basic Structure

This basic diagram can be extended onto complex relationship problems, where inference of the network relationships is required.

The strength of connections between the events is measured in terms of probabilities, with each node having an associated conditional probability. The node then allows for calculation of the outcome of the child event given its possible combination of parent outcomes.

5.2.2 Inference

As outlined in AI for Game Developer by Bourg and Seemann (2004, 247-248), there exist 3 basic types of inference network layouts:

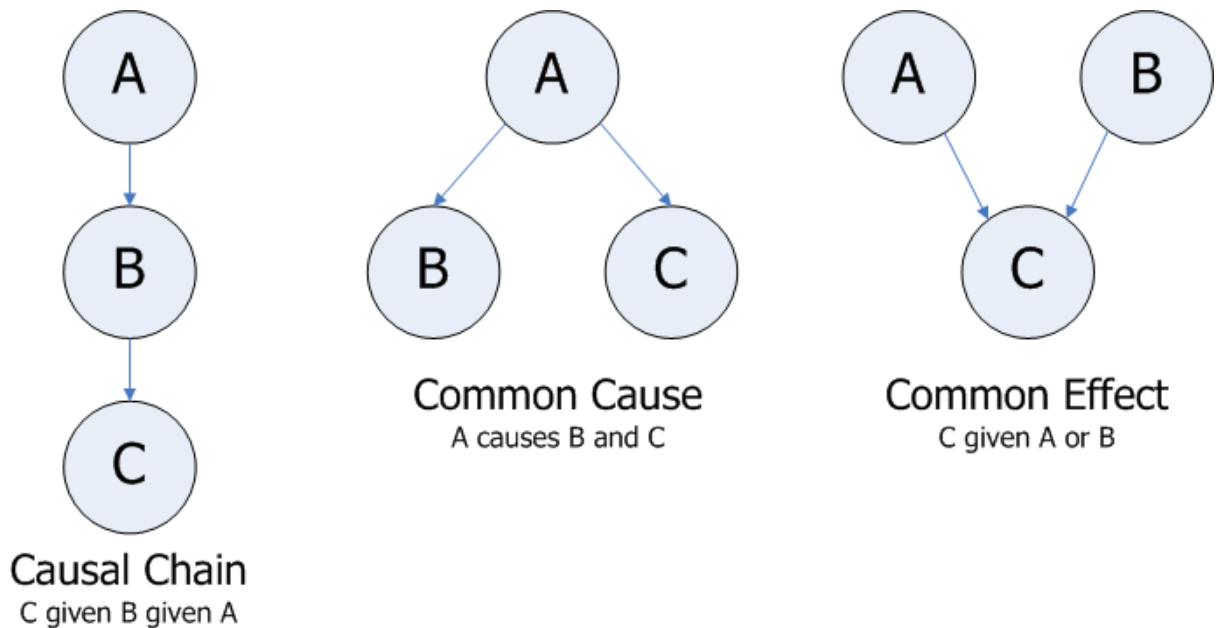


Figure 5-2: Simple Networks

These three basic layouts can be used as a basis for three basic types of reasoning, Predictive, Diagnostic and Explaining respectively.

5.2.2.1 Predictive (Causal Chain)

Predictive reasoning is a forward logic analysis of the Bayesian Network, where decisions about effects are based on knowledge of causes. Such as if we know of B, which causes C, we can then calculate the probability of C.

5.2.2.2 Diagnostic (Common Cause)

Diagnostic reasoning is a process which is commonly used in medical diagnostics. As Bourg and Seeman state in AI for Game Developers (2004, 248), referring to the common cause network, if A was a disease and B and C are symptoms, then a doctor could make inferences as the probability of the disease being present, based on the existence of B and C.

5.2.2.3 Explaining (Common Effect)

Explaining is the process of inference when an effect is common to multiple causes. Such as if we know that C is true, we can create probability that A and/or B are true, as they cause C.

5.2.3 Implementation

Bayesian Networks are utilised in many game AI systems. One example is that of a fighting-game. The computer opponents are required to predict the next strike that the player will throw and take appropriate actions. A Bayesian Network system is used to track the probabilities that a player will execute a particular action, based on past behaviour – however, the player may choose a different attack, so a level of uncertainty exists.

A Bayesian Network can be generalised to allow for a computer character to decide their next decision, based on desired effects or causes. This formation of a decision network can form the base of a goal-based AI system, as outlined in Section 4, Goal-Based Techniques.

Bayesian Network analysis is utilised not only within game AI systems, but in decision making processes of other applications.

5.2.3.1 Lumiere

Lumiere is a Microsoft project where the aim was to create a software product which would detect the needs of the user of the system. It was based on earlier research into pilot-aircraft interaction systems, where relevant information to the current situation is highlighted. The Microsoft Lumiere project resulted in the creation of the Office Assistant, which has been included in Microsoft Office since Office '95.

5.2.4 Limitations

Although Bayesian Networks are an excellent tool for inferential processes, there are some inherent limitations. As they rely on some knowledge to calculate the initial probabilities, the applicability of the network to describe the process, can only be ever be as effective as the reliability of the knowledge used to calculate it; probabilities used to describe the situations must be correct.

The main limitation of using a Bayesian Network in game AI routines is the potential difficulties in calculating an unexplored network. As described by Niedermayer in *An Introduction to Bayesian Networks and Their Contemporary Application*

To calculate the probability of any branch of the network, all branches must be calculated. While the resulting ability to describe the network can be performed in linear time, this process of network discovery is an NP-hard task which might either be too costly to perform, or impossible given the number and combination of variables. (1998)

It highlights the need to pre-traverse common networks to reduce processing time and ensure that the networks in use can be calculated in a reasonable time.

6 A* PATH FINDING

6.1 Description

The A* algorithm is a tried and tested solution designed for the problem of plotting the shortest path from start to finish point. Whilst there are probably many different solutions to this common problem, what sets A* (pronounced A star) apart, is that it can find the shortest path relatively quickly. Used since 1968, the algorithm according to Brian Stout in *Game Programming Gems* (2000, 254), searches in a "*state space* for the least costly path from start state to a goal state by examining the neighbouring or adjacent states of particular states.". This means that the algorithm will go through examining each node and then its adjacent nodes and repeat this until the quickest path from start to finish can be found. It is known as a "directed" algorithm, as it uses an educated guess type approach to find its final goal, rather than just searching blindly. It will even backtrack to find alternative means, making the A* algorithm a flexible solution to the now, age-old path finding problem.

6.2 Implementation Details

6.2.1 Overview

The theory behind the A* algorithm is what drives the algorithm to be so successful, widely used and thus important within game AI. A* utilizes nodes, which are basically a location on a map. Imagine the node, in terms of a standard Cartesian plane graph, with X and Y axis, as being a point plotted at say A(2,3) for example. This point, A, is a node within the A* algorithm and each of these nodes has to have various properties associated with it.

6.2.2 Properties of Each Node

Each node within the A* algorithm has to have a certain amount of properties associated with it in order for the algorithm to function successfully. The attributes of each node required in the A* algorithm include:

- Actual Node Location coordinates,
- Goal (aka g , $g(x)$ or Cost from start),
- Heuristic (aka h , $h(x)$ or Cost to goal),
- Fitness (aka f , $f(x)$ or Total path) and a
- Pointer to previous Node location

Of course the node must have information of where on the map it actually exists, its actual location. Along with this important piece of information, a node must also have a pointer to its previous location – an adjacent node – one that it was at prior to arriving to its current location. Three other extremely important figures are associated with each individual node, namely Goal, Heuristic and Fitness.

6.2.3 Goal

The goal or g value within a node, using the A* algorithm, is the cost to get from the starting point to the current node. Basically this means that it is the distance or nodes travelled so far from the original starting node. Many different paths may in

fact be taken to return to the starting point from the current node, but the g value that is used is the one for the specific single path being investigated. As the algorithm has explored all previous nodes that led to this one, the g value therefore is a figure which can be calculated specifically by totalling all previous nodes to the start node.

6.2.4 Heuristic

The heuristic or h attribute is where the educated guess comes in to play within the A* algorithm. As the distance from the current node to the final destination is not yet known (i.e. what we are trying to discover), an educated guess or heuristic is required. It is an estimated movement cost from the current node to the final goal destination. Many implementations use various techniques in order to produce this estimate, however probably the most common is the Manhattan distance technique. Thus this figure can be obtained by utilizing the Manhattan distance technique in most cases which will provide an effective estimate of the distance to the destination node.

6.2.5 Fitness

The fitness attribute f , despite its odd name, is the attribute used to decide whether or not to pursue a specific path. In essence, it is the total path estimate – a figure showing the approximated distance from the start node to finish node, via the current node and any predetermined previous nodes. The A* algorithm uses this information to make informed decisions upon which route to the goal is the shortest and hence most effective. The figure is achieved by simply adding the heuristic and the goal i.e. $f = g + h$. Of course the lower that this figure is, the better and shorter the path to the destination via the current node will be. Also of note is that the better the heuristic is in approximating the gap between the current node and the destination will in turn mean the closer f is to its true value and therefore will ensure limited wasted effort by the algorithm.

6.2.6 Other A* Components

Along with the nodes and their attributes the A* algorithm also maintains two lists, known as *Open* and *Closed*. James Matthews, in *AI Game Programming Wisdom* states that, "The open list is used for listing nodes that *have not* yet been fully explored. The closed on the other hand is used for listing nodes that *have* been explored." (2000, 106) These lists are required therefore by the algorithm in order to track which nodes have been looked at and which have not to avoid examining the same nodes twice.

Of course a node can only be fully considered to have been explored (and hence added to the Closed list) once all it's adjacent nodes have been added to the Open List and their attributes calculated.

6.3 Pseudo Code

The A* algorithm, utilizing the previously mentioned attributes and components that it is provided with can then set about finding the shortest path from start to finish. In order to do this, the attributes are used and evaluated in a certain fashion. The following pseudocode outlines this fashion and is referenced from the "Basic A* Pathfinding Made Simple" article by James Matthews, in *AI Game Programming Wisdom* (2000, 107) and most clearly defines the operations that are carried out by the A* Algorithm: -

1. Let P = the starting point.
2. Assign f, g, h values to P .
3. Add P to the Open list. At this point, P is the only node in the Open list.
4. Let B = the best node from the Open list (i.e. best node has lowest f -value)
 - a. If B is the goal node, then quit – a path has been found
 - b. If the Open list is empty then quit – a path cannot be found.
5. Let C = a valid node connected to B .
 - a. Assign f, g and h values to C .
 - b. Check whether C is on the Open or Closed list.
 - i. If so check whether new path is more efficient (lower f -value)
 1. If so, update the path.
 - ii. Else, add C to the Open list.
 - c. Repeat Step 5 for all valid children of B .
6. Repeat from Step 4.

6.4 Weaknesses or Limitations

Although A^* is one amongst some of the best search algorithms available, it still has its weaknesses and limitations. The area in which A^* is most inefficient according to Bryan Stout in Game Programming Gems (2000, 261) is, "in determining that no path is possible between the start and goal locations; in that case, it examines every possible location accessible from the start before determining that the goal is not among them". This obviously would have repercussions within games, if the A^* path finding technique was not able to provide a path for the agents to follow, and thus they could not perform a required action. On top of this it would also utilize a large amount of resources whilst trying to discover the unavailable path. There is however a method to avoid this occurring which involves performing a pre-analysis of the map and performing the search only if a path is achievable.

7 ACRONYM GLOSSARY

Item	Description
AI	Artificial Intelligence
NPC	Non player character
RPG	Role playing game
FSM	Finite state machine
RTS	Real time strategy
A*	A Star search algorithm
NOLF2	No One Lives Forever 2

8 REFERENCES

- Bourg, D., G. Seemann. 2004. *AI for Game Developers*. Sebastopol: O'Reilly Media, Inc.
- DeLoura, M. ed. 2000. *Game Programming Gems*. Hingham: Charles River Media, Inc.
- Finney, K. 2005. *Advanced 3D Game Programming All In One*. Boston: Thomson Course Technology.
- Niedermayer, D. 1998. *An Introduction to Bayesian Networks and their Contemporary Applications*. <http://www.niedermayer.ca/papers/bayesian/> (accessed April 5, 2006).
- *No One Lives Forever 2 AI Q&A*. 2002. <http://actionvault.ign.com/features/den/nolf2qa01.shtml> (accessed April 3, 2006).
- Rabin, S. ed. 2002. *AI Game Programming Wisdom*. Hingham: Charles River Media, Inc.
- Sanchez-Crespo, D. 2003. *Core Techniques and Algorithms in Game Programming*. Indianapolis: New Riders.
- *The American Heritage Dictionary of the English Language, Fourth Edition*. 2000. Boston: Houghton Mifflin Company.
- *The Elder Scrolls IV: Oblivion*. 2006. http://www.bethsoft.com/games/games_oblivion.html (accessed April 3, 2006).