REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-04-

0278

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
15-05-2004	Final Technical Report	15 AUG 2003 - 15 MAY 2004

4. TITLE AND SUBTITLE
Adaptive Artificial Intelligence for Next Generation Conflict

5a. CONTRACT NUMBER
F49620-03-C-0062

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
John Tiller
John Rushing
Drew McDowell
Steve Tanner

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
John Tiller
142 Sarah Hughes Dr
Madison, AL 35758

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
4015 Wilson Boulevard Room 713
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Report developed under STTR contract for topic "Adaptive Artificial Intelligence for Next Generation Conflict". This project addresses an architecture and development approach for Artificial Intelligence in computer wargames. The results of this project were used in three technology test-bed applications having to do with modern air power, squad-level ground combat, and grand-operational warfare. The AI architecture was designed so that potential AI technologies can be utilized in this architecture in a "plug-and-play" manner. Under this architecture, new AI developments will be easy to incorporate into the AI engine. This will enable the resulting computer wargames to remain current and competitive in the future.

15. SUBJECT TERMS
STTR Report, Artificial Intelligence, Wargame

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			John Tiller
None	None	None	UU	72	19b. TELEPHONE NUMBER (Include area code) 256-461-8652

Phase I Final Report

Submitted by: John Tiller
tiller@hiwaay.net
256-461-8652

For: STTR AF03T022

Adaptive Artificial Intelligence for Next-Generation Conflict

Approved for public release; distribution unlimited.

Table of Contents

1. Introduction

This final report covers the work by John Tiller and the University of Huntsville-Alabama for the Phase I effort under the Air Force Office of Scientific Research funded STTR AF03T022 for the period August 15, 2003 through May 15, 2004.

The research conducted under this effort was based on two primary directions: the development of technology test-bed applications that would serve as development platforms in a subsequent Phase II and the investigation into AI technologies that would be used in AI development in Phase II.

For this purpose, three technology test-bed applications were eventually used:

- Modern Air Power – a real-time simulation of theater-level air power.
- Total War in Europe – a grand-operational simulation of World War II in Europe.

- Squad Battles – a turn-based tactical ground combat simulation.

In each case, preliminary AI development was conducted to serve as a basis for subsequent development in Phase II. In the case of Modern Air Power, this was sufficient to bring the first commercialization War Over Vietnam to completion for release by mid-May 2004.

2. Technical Results

The research and development pursued under this effort followed the approach first described in the Phase I proposal. This approach structures the AI according to a three phase system of Assessment, Planning, and Execution. Figure 1 below shows how this approach is structured relative to the game engine and the AI technologies.

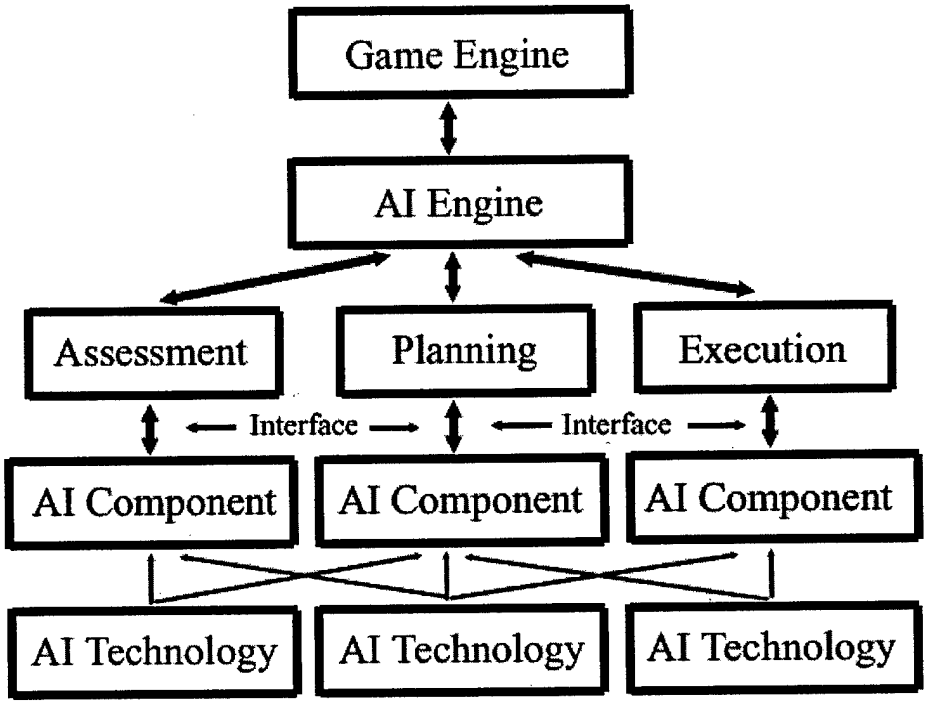


Figure 1. A/I System Architecture

These three phases of the AI engine are described below:

1. **Assessment** – Determine the key issues with respect to the current situation. Incorporate this with previous information so as to be able to identify trends.

2. **Planning** – Attempt to predict enemy Course of Action. At the same time, conduct planning as to friendly Course of Action. Both of these rely on assessment information gathered in the first step.
3. **Execution** – Execute the plans developed in the previous step. Determine optimal implementations of these plans using assessment information.

This structure allows development of the AI technologies to proceed independently of the game engine development and allows for flexibility in the application of new AI technologies into the game system. Based on this approach, it was decided that the research institute, the University of Alabama-Huntsville, would take responsibility for development of an AI library based on several different AI technologies while the principle developer, John Tiller, would proceed with development of the game engines. The architecture described here then allows the linkage of those game engines with the resulting AI library in subsequent Phase II development.

The AI technologies investigated during this Phase I effort include the following:

- **Expert Systems** – rule-based systems capable of performing logical reasoning.
- **Neural Networks** – general input/output functions particularly suitable for non-linear systems.
- **Genetic Algorithms** – optimization techniques that are robust in non-linear situations.
- **Cluster Analysis** – useful for extracting data from complex systems.
- **Data Mining** – potentially useful in education situations for extracting performance data.

Similar to this AI approach, it was also decided that development of Cascading Effects could proceed in a similar manner. That is, it was decided that the research institute would proceed with the development of a general Cascading Effects engine and at a suitable point in its development, the principle developer would link the game engines to that code.

Much of the technical research conducted in this Phase I effort is described in more detail in the Reports section of this document.

3. Conferences and Presentations

The following two subsections list the conferences and presentations made during this STTR and also planned conferences under the Phase II effort to be subsequently performed.

Actual

- JT: Two presentations on Computer Wargames at 2003 Connections, plus Modern Air Power demo, Rome AFRL, July 15-18th, 2003.
- JT: Presented Modern Air Power demo at Maxwell AFB, September 26th, 2003.

- JT: Attended DARWARS demo at DARPA and presented Modern Air Power demo, October 16th, 2003.
- JT: Met with Stottler-Henke developer Dan Fu, October 22nd, 2003.
- JT: Attended Training Games Symposium, DC, November 5-6th, 2003.
- JT: Presented seminar "Implementing AI Technologies in Computer Wargames", University of Louisville, and met with Intellas developers, November 7th, 2003.
- JT: Attended I/ITSEC convention, Orlando, FL, Dec 1–4, 2003.
- JT: Modern Air Power research and design at Maxwell AFB, Jan 22-23, 2004.
- JT: Attended JFAC exercise at Maxwell AFB, Feb 17-18, 2004.
- JT: Attended AF M&S conference, Orlando, FL, Feb 24-26, 2004 and served on AF wargaming panel. Appreciation letter from Col. Votipka.
- JT: Presented at the "Defense & Security Symposium", April 12-16, 2004, Orlando, Florida.
- JT: Presented at the AMC Technology Days Symposium, May 5-6, 2004, Scott AFB, Illinois.

Planned

- JT: Attend AETC Conference and present, May 25-27, 2004.
- JT/UAH: Attend AI Conference, San Jose, July, 2004.
- JT/UAH: Attend Connections Conference, USAFA, August, 2004.

4. Summary

The Phase I effort of this STTR has brought 3 wargames forward in their development that will be useful to the Air Force for education, training, and simulation. The Modern Air Power game series has started commercialization and there are an additional 3 or more games planned in this series, plus the military professional version. The Squad Battles game series will be further developed to address current and future weapons and technologies associated with Air Force Security Force situations. And finally, the Total War series will be developed to completion so that it can be used to look at strategic issues with cascading effects in both historical and modern conflict.

5. Reports

The following sections describe in more detail development that was accomplished in the Phase I effort of this proposal.

5.1 Wargaming for Blue Victory

Matthew Caffrey, Col, USAFR
John Tiller, Ph. D., Commercial Wargame Developer

Introduction

It was after the attack on Pearl Harbor during World War II and the Japanese were making preparations for their next big campaign, that of the naval attack on the American-held Midway Island. In preparation for this as was their custom, a wargame was held to rehearse the attack, just as had been done earlier for the attack on Pearl Harbor. But things were not going well for the Japanese side in this wargame. During the game, American planes sank two Japanese carriers. Rear Admiral Ukagi Matome, commander for the Japanese force for the future operation unilaterally reversed the judgment of the umpires and restored the carriers to the game. Restarting the game, the Japanese went on to capture Midway Island. However, two weeks later in the actual operation, American planes did sink two Japanese carriers, and then two more. Unfortunately for the Japanese, Admiral Matome could not reach into the "dead pile" and replace his ships.

This paper presents a concept called "Wargaming for Blue Victory" and discusses the issues associated with this. The Japanese conduct of their Midway wargame is a very good example of both the concept and its potential consequences. The conclusion of this paper is that to avoid situations such as these in future military conflicts, a strong computer opponent with capable, challenging Artificial Intelligence (AI) is needed.

Background

Until recently, wargaming was based entirely on human adjudicated games. That is, given two sides Red and Blue engaged in a wargame scenario, both sides were required to be played by human opponents and the resolution of the wargame rules, or adjudication, was also required to be managed by a human judge. Before the introduction of computers, this was the only approach available. As with any endeavor involving human participants, there is the danger that subjective human judgment may have an adverse effect on the course of the activity.

In the early 1800's, as wargaming, or "Kriegspiel", was coming of age in Prussia, General Moltke would conduct exercises with the Prussian War College at one of the actual invasion corridors into Prussia. Perhaps realizing how human factors can adversely influence judgment, Moltke would begin by turning to the most junior student present and ask for his plan of battle. Next he would ask the same question of the second most junior student, and so on up the seniority chain. In doing so, he effectively neutralized the very real tendency of any junior officer being placed in a situation of having to contradict his senior.

Moltke certainly recognized the problem called today "command influence". In any military situation, the judgment of more senior officers is going to overrule, either overtly or covertly, the otherwise natural decisions of the junior officers. In wargaming exercises, this can have an adverse effect on the objective outcome of the game. For example, in the late 1800's, Kaiser Wilhelm II, a self-considered military expert, would also conduct staff rides to the invasion corridors. However, upon arriving, the Kaiser would immediately announce to all present what he considered to be the "perfect" battle plan. Not surprisingly, when the wargame was carried out, the Kaiser's side always won and the Kaiser's preconceived notion vindicated. In the Great War to follow, this

weakness in Germany military planning would have a disastrous effect in the invasion of France.

Concept
In this paper, the concept of "Wargaming for Blue Victory" has this definition:

> **Wargaming for Blue Victory** means a situation, process, or effort, either deliberate or inadvertent, causing the outcome of affected wargame scenarios to be biased towards friendly victory.

Wargaming for Blue Victory skews the otherwise meaningful results of wargames so that the participants or reviewers of the outcome end up with a false impression of the relative difficulty of a particular situation or more importantly, are often unprepared for serious issues and consequences associated with the actual situation.

Causes
There are several reasons why Wargaming for Blue Victory can occur in wargaming. The historical situation caused by the Kaiser in Germany's wargames in the late 1800's, is one good example of how command influence can result in this effect. The comments by the Kaiser prior to the conduct of the wargame established a predisposition towards the eventual outcome that was impossible to overcome.

The construction of the wargame rules can also result in a bias towards Blue victory. During World War II, the US Army held wargames based on new equipment and technology, much of which was unfamiliar and untested in battle. The head of the tank-destroyer program provided the adjudication guide for the effectiveness of the tank destroyers in the wargame. Later events would show that these guides overstated the lethality of tank-destroyers. In early battles, tank destroyers were used far too aggressively and with tragic results.

In these same American wargames, efforts were made before play began to guarantee that the outcome would provide the ground officers' position on the employment of airpower. It was until later after disasters like the Battle at Kasserine Pass that procedures were changed.

In the traditional conduct of wargames with human players for both sides, it is possible for the allocation of participants to be biased towards Blue. In the German wargames, the most senior officers were given the privilege of serving with the Kaiser on the German side while the junior officers were given the task of performing the function of the opposing side. This imbalance in skill and experience certainly helped contribute to the one-sided outcomes.

And certainly in this situation, it would be human nature for the players of the opposing side to avoid taking actions which would embarrass the friendly side or show some weakness in the friendly side. Regardless of rank, if the Red side is being played by Blue officers, they can, if only covertly, have a bias towards Blue victory.

Finally, as the Japanese wargame of Midway Island shows, whenever the participants have the option of overriding the results of the wargame as it progresses, then there is the very real possibility that all objectivity is being lost in the outcome of the wargame and that the result can totally misrepresent the actual scenario.

Prevention

The prevention of Wargaming for Blue Victory can occur through the use of computer-based, rather than human-based, wargames, with the objective application of analytical rules, and the introduction of a competent and challenging Artificial Intelligence feature for the conduct of the opposing side.

While it is not impossible that the programming of a computer wargame can introduce a bias towards friendly victory, the process of computer coding requires that the rules be formalized and implemented in a strict computer language. Such an approach results in source code that can be examined by others and any bias introduced into the coding can be identified. Likewise, the execution of these rules is then performed objectively by the computer hardware and this prevents any human subjective resolution, however slight or inadvertent.

And while it is not impossible that the players in a computer wargame stop the wargame and restart it to avoid adverse outcomes, it becomes a very explicit act in the conduct of the wargame rather that what could be considered the honest correction of a mistake in adjudication.

The introduction of Artificial Intelligence for the Red player in wargames removes any potential for command influence in the execution of the game. Likewise, an AI player has no problem with embarrassing the Blue player or demonstrating the fallacy of the Blue course of action. And an AI player is not at risk of destroying their career by doing so.

Benefits

The approach of basing the conduct of wargames on computer implementations with competent AI opponents has several benefits. The first of which is the mitigation, if not elimination, of the Wargame for Blue Victory syndrome. In addition, with an AI opponent, it is then possible to introduce programming which implements national characteristics. When this is done, the Blue player no longer can count on seeing typical Blue responses from the Red opponent during the game, but must plan and react more according to the natural national characteristics of a real Red opponent.

There are significant cost benefits to using computer-based wargaming with AI opponents as well. Typical live wargame exercises can involve hundreds of military personnel incurring significant costs for the duration of the exercise. The present day cost of computer equipment is so small in comparison that a huge increase in the number of wargame experiences and their frequency is possible.

Finally, the resulting wargame is of high value as well because the automation of the adjudication and the Red decision process then results in wargames with a much tighter so-called OODA loop. That is, the resulting loop of Colonel Boyd of "Observe, Orient, Decide, and Act" is operating at a much faster rate for such a wargame, resulting in meaningful results in a much shorter time. This can be invaluable when in a situation requiring the fast turnaround of course of action evaluation.

"A good plan today, viperously implemented, is better then a perfect plan tomorrow."
General George S. Patton Jr.

Conclusion
Using computer wargames with challenging Artificial Intelligence, we are much less likely to have wargame results skewed towards Blue victory and thus we will benefit far more from these results than we would otherwise do with human-based adjudication and human opponents. It is in our best interests to have the best anticipation of future conflict and the ability to evaluate and make decisions faster than we have done in the past.

"This is not the enemy we wargamed against."
Lt Gen William Scott Wallis
CG V Corps, Operation Iraqi Freedom

5.2 The Determination and Application of a Concept of Frontline in Ground-Based Operational-Level Wargames

John Tiller

1.0 Introduction
This report addresses the concept of **frontline** in ground-based operational-level wargames. The scale of operational-level is defined as wargame situations where there is no significant ranged-fire and thus the influence of any particular ground unit consists of the area that it occupies but not to any far ranging distance from that area. It would be possible to introduce ranged-fire into the algorithms through enhancement and extend the notion of frontline given here to more tactical wargames, but this report will focus on operational-level and strategic-level wargames.

The resulting concept of frontline will then be applied to a significant example of a ground-based operational-level wargame. This wargame is the **World War II in Europe** wargame series developed by John Tiller for HPS Simulations. This series has 5 potential releases beginning with The First Blitzkrieg covering the initial campaigns of the Second World War in Europe such as Poland, France (1940), and Norway.

2.0 Definition
The concept of "frontline" is intended to provide useful information to AI (Artificial Intelligence) algorithms in computer wargames. To begin with, a definition of frontline

needs to be established before algorithms that generate this result can be developed. In this report then, the concept has this definition:

*With respect to a given side in a wargame scenario, the **frontline** of that side is defined to be the boundary of the area of influence for that particular side.*

It should also be noted that once the frontline boundary is determined, this is then the starting point for subsequent AI algorithms based on that frontline. For example, it can then be determined which locations on the frontline are secure and which are in danger. Or correspondingly, it can be determined which locations on the frontline have potential for friendly exploitation. If an objective has been defined as a potential future frontline for a given side, then that can be used to determine if the objective has been reached or what locations have not been attained by the friendly side.

Also notice that the concept of frontline is not symmetric. That is, the frontline for one particular side can be different from, but not overlapping with, the frontline of the other side. In a sense, the intervening area between the frontlines is a type of *no-man's land* with respect to a particular situation, or perhaps more appropriately for operational-level situations, an area of disputed influence.

3.0 Development

To develop algorithms that result in the determination of frontline, there needs to be a determination of what constitutes **influence** in a particular situation. To start with, each friendly unit exerts influence in the location it occupies. Given our definition of operational-level scale, this natural influence extends no further. Any extension of the area of influence for the friendly side will then depend on the existence and location of enemy units.

3.1 Danger Metric

The first step then towards the establishment of area of influence is to determine not only where enemy units exist, but also the extent of their threat to a given location on the wargame map. To determine this, we introduce the notion of a **danger** metric. This has the following definition.

*The **danger** value of any particular location is a value related to the ability of enemy units to occupy that location.*

For example, for locations now occupied by enemy units, the danger value is highest. For locations further away from enemy units, the danger value decreases. It is the comparison between two given danger values that is of primary use to us rather than the specific danger value.

An algorithm to generate danger values is easy to construct by first assigning enemy locations a high danger value, and then in successive passes, assign lower danger values to adjacent locations until all locations have been considered.

3.2 Area of Influence

Using the location of friendly units and the danger values determined, it is then possible to define an area of influence for a given side. Note that this definition is not objectively determined as it depends on the decision of whether locations at a distance from friendly units are still under the influence of friendly units.

To begin with, all locations occupied by friendly units are considered under the influence of friendly units. We will assign these locations a working value of the danger to use in the algorithm.

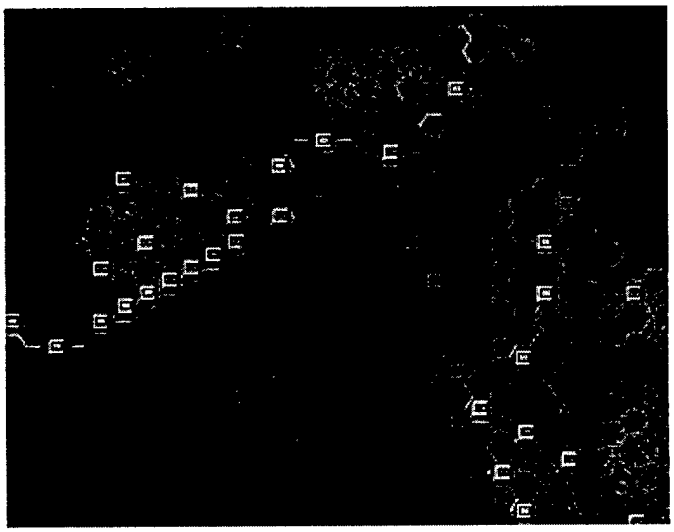
Next, for each location that has been assigned a working value, we take that value and increase it by a certain amount. The amount of this increase determines the sensitivity of the algorithm to the presence of enemy units. For each adjacent location that has not been assigned a value, if the modified value is greater than or equal to the danger value at that location, we assign that location a working value equal to the modified value.

The algorithm continues until all locations have been considered. At this point, the area of influence is defined to be all locations that have been assigned a working value by the algorithm. From this, we determine the frontline to be the boundary of the area of influence.

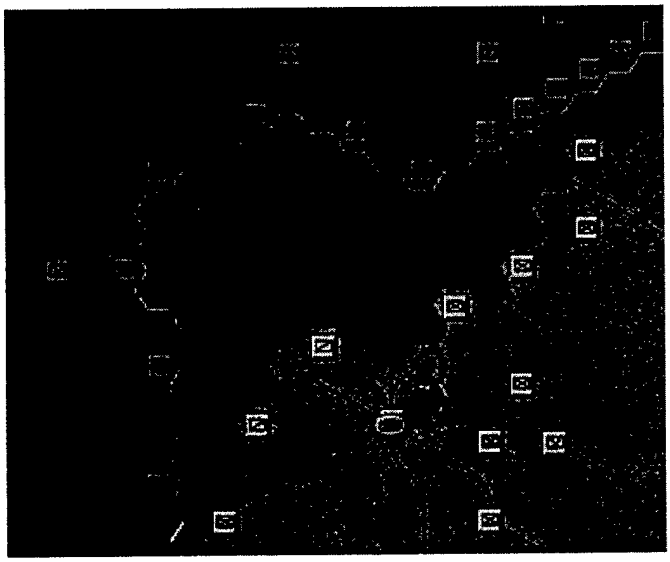
4.0 Results

Below are snapshots showing the area of influence determined by this algorithm in the computer wargame **The First Blitzkrieg**, currently under development by John Tiller for HPS Simulations:

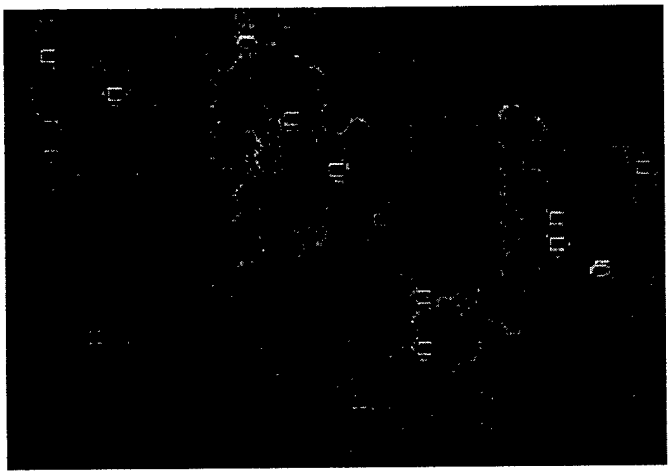
German Frontline: Note that depending on how close the enemy units are, the algorithm does not extend the area of influence between friendly units.



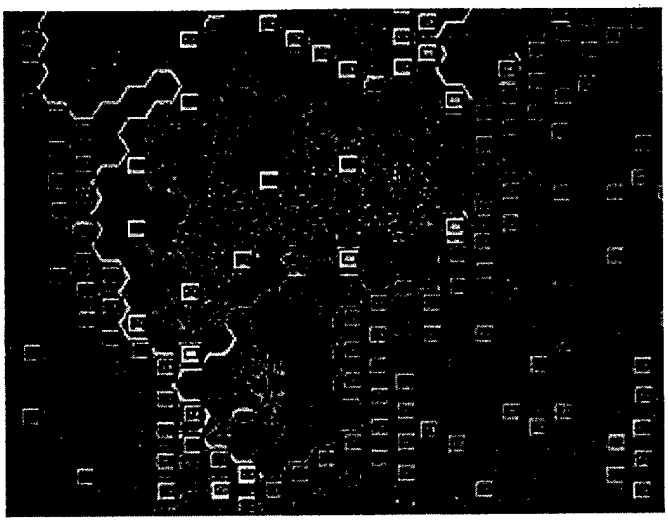
Polish Frontline: Notice that the algorithm identifies significant areas of the Polish frontier as failing to be under Polish influence.



Danish Frontline: This is after the first German turn and as a result of the German invasion, large portions of Denmark are identified as no longer being under Danish influence.



Allied Frontline: This is on the first turn and several issues with respect to the area of influence of the Allied side have already been identified by the algorithm.



5.3 Code Structure for the Implementation of AI Programming in the Model-View-Controller Design Pattern

John Tiller

1.0 Introduction

This report discusses an approach to code structure in computer wargames that supports a complete implementation of the **Evaluate-Plan-Execute** (EPE) AI logic loop within the context of a **Model-View-Controller** design pattern. This approach then allows for a completely flexible implementation of the EPE loop, implementation of flexible plug-and-play AI technologies, portability to other wargame implementations, and flexibility in the EPE loop implementation.

2.0 The Model-View-Controller Design Pattern

The Model-View-Controller (MVC) design pattern is an approach to structuring the code of an application into three distinct parts each with its own unique responsibilities. In this approach, the **Model** component of the application is responsible for the low-level engine calculations associated with the application. That is, the Model component is designed to be an abstract implementation of the primary functions of the application and should be completely free of any references to user interface or graphics. The **View** component of the application encapsulates all of the graphical processing of the application and all functions associated with the graphical representation of application functions. Finally, the **Controller** component of the application contains all user-interface functions such as processing button clicks, dialog processing, and other actions initiated by the user through the interface.

The call structure within the MVC design pattern is significant. The Model component only makes calls to itself and never directly calls the View or Controller components. In this way, it is programmed independent of any particular graphical or user interface implementation. The View component can make direct calls to the Model, but only for

the purpose of obtaining information, but not to make any changes to the Model. The View is thus programmed simply as a means of representing the Model data and not as a means of modifying or updating that data. Finally, the Controller makes calls to both the Model and View components. In both cases, it has the ability to make changes to either component. Typically, as a result of user interaction, the Controller calls the Model to cause some change in the state of the data and then calls the View component to cause any representation of that data to be updated.

3.0 The Evaluate-Plan-Execute AI Processing Loop

The Evaluate-Plan-Execute (EPE) AI loop is based on a three-stage approach to AI processing:

1. **Evaluate** – Evaluate the current situation.
2. **Plan** – Generate plans on how to handle the current situation.
3. **Execute** – Execute those plans.

The desirable approach is to encapsulate that processing into a single component so that the maximum flexibility is achieved in the implementation. As such, the EPE AI loop represents a portion of Model processing in the MVC design pattern. It has access to other Model data and can modify that data, but relies on the other components of the MVC design pattern for representation of those changes.

4.0 The Integration of the EPE Loop within the MVC Pattern

As previously stated, the EPE AI loop should be viewed as a portion of the Model component. In this approach, the Controller component is then responsible for invoking the EPE loop as needed, either directly or indirectly through other calls to the Model, so that AI processing is performed. However, in this approach, there is a lack of visibility into AI processing as a single call to the EPE loop would result in the entire AI processing of the loop being performed without any intermediate visualization of the results. An additional feature, that of **callbacks**, needs to be added to this approach to achieve this visualization.

4.1 Callbacks

A callback allows an entity to trigger abstract events without having to know the recipient of the trigger. This notion is completely similar to the notion in computer hardware of *interrupt*. A hardware interrupt is generated by computer hardware and this interrupt is then processed by driver or operating system software. This approach to hardware and software design then allows hardware to be designed and built independent of the operation system software that will use the hardware. A software callback acts in a similar way as a trigger for further processing without the knowledge of what exactly that processing will entail. It is the responsibility of the entity that triggers the callback to make available whatever information might be necessary by the code that processes the callback just as computer hardware that generates interrupts needs to have information available in hardware registers for processing by the driver or operating system.

A software callback is therefore an abstract class with a single pure virtual function member that we can call **Invoke**. When a higher-level software entity registers a

callback with a lower-level entity, it then provides that lower-level entity with a derived class to the base class of callback, one that will invoke a member function specific to that higher-level entity.

This approach allows the lower-level entity to perform processing, invoke the callback to have that processing recognized, and then possibly continue processing. This approach thus allows the EPE loop to be written as a single code entity with callbacks at appropriate points within that processing so that the higher-level Controller and View components can generate representations of that processing.

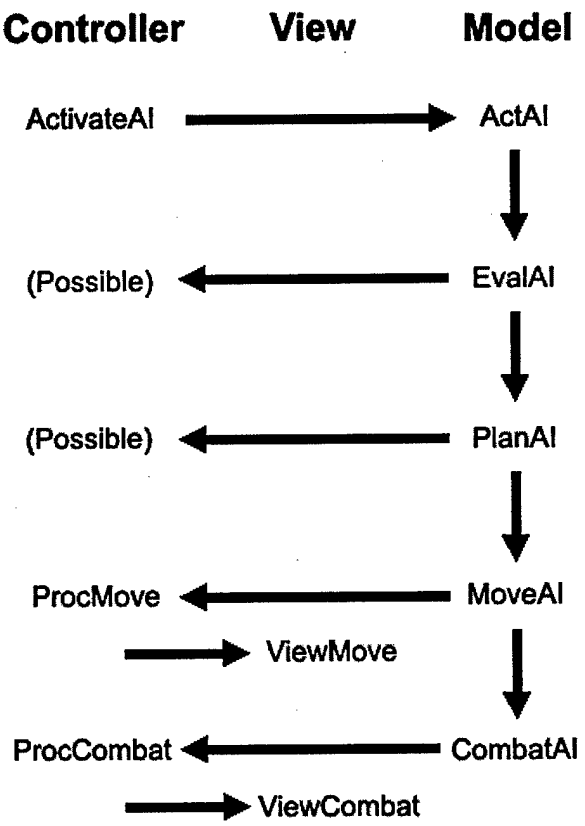
4.2 The Resulting Structure in a Wargame

Implementing this approach in a turn-based computer wargame results in the following structure and possible callbacks:

1. Within the Evaluate section, a possible callback so that results of the evaluation can be viewed within the application.
2. Within the Plan section, a possible callback so that results of the planning can be viewed.
3. Within the Execution section, possible callbacks for the actions of movement and combat so that individual steps in this processing can be viewed in the applications.

5.0 Conclusion

The resulting code structure and call sequence can be visualized in the diagram below.



This approach allows a complete EPE AI loop to be coded independent of the graphical or user interface of the application. This then allows the full benefits of an EPE loop approach to be realized.

5.4 A Parameterized Minpath Algorithm with Applications to Computer Wargames

John Tiller

1.0 Introduction

This report addresses the design and implementation of a parameterized minpath algorithm that is particularly suited for use in computer wargames. There is a high need for a generalized minpath algorithm in computer wargames. This algorithm has two important applications:

1. Shortest distance path finding to facilitate user movement.
2. Shortest distance path finding for use by the AI.

In most cases, the distance computation that is used for this path finding can be quite involved and depend on particular issues of the particular wargame engine. In addition, it

is not uncommon that the distance computation be based on subjective factors. For example, in a path calculation that potentially could involve traversing a minefield, how "costly" should the algorithm consider the casualties resulting from moving through the minefield as compared with the extra distance involved in moving around it. This determination can be based on **national characteristics**. That is, for Russian units, the casualty cost could be considered quite low while for American units, it could be considered quite high. Based on this, the algorithm could determine that the "shortest" path was either through the minefield or around it.

2.0 Framework

In order to efficiently implement the traditional Dijkstra Minpath Algorithm in a computer algorithm, it is important to recognize that many computer wargame maps are quite large while the movement path that results may only involve a small fraction of that map. For this reason, it is impractical to consider constructing a traditional pre-defined graph with nodes, edges, and costs. Therefore, to achieve an efficient implementation, it is necessary to define an abstract **Dynamic Graph**. A Dynamic Graph is based on an abstract notion of node and has three member functions defined:

- The **cost** of moving between any two nodes in the graph.
- For each node, a list of **adjacent nodes** that have not been "marked".
- For any node, the ability to **mark** that node and thus consider it part of the search tree.

When the minpath algorithm resulting in this report is implemented, then a definition of Dynamic Graph must be provided by the application and these three member functions must be implemented.

3.0 Implementation

Based on this definition of Dynamic Graph, the traditional Dijkstra Minpath Algorithm has a straightforward implementation. To construct the minpath between two given nodes **a** and **b**:

- Begin the construction of a tree by including **a**.
- While not done, consider all nodes **adjacent** to nodes in the tree, determine the **cost** of including each of those nodes in the tree, and pick one of minimum cost.
- As each node is included in the tree, **mark** it so that only nodes not in the tree are included in subsequent iterations of the algorithm.
- Stop once node **b** has been included or no additional nodes exist to include.
- Starting from the last node included, build the resulting path backwards from node to previous node in the tree until you reach node **a**.

4.0 Specializations

Once the algorithm is established, it is possible to vary the results by specializing the cost function. As previously mentioned, one issue in the definition of the cost function is the determination of "intangible" costs such as minefields. Likewise, another determination is whether to include the presence of enemy units in the resulting cost. That is, if enemy units are ignored, the units which are attacking will "charge" the enemy units and confront them. If enemy units are included in the cost function, then the algorithm may

cause the attacking units to bypass the enemy, resulting in unsatisfactory results in the game.

Another consideration in the definition of the cost function is the extent to which "detours" are considered. For example, consider the problem of finding the shortest path across a river. There may be one bridge adjacent to the current location and a second a large distance away. If friendly units are currently using the bridge and it cannot carry any additional units, then a strict implementation of the shortest path algorithm would result in the waiting units moving to the second bridge, a very large detour. This solution may be completely unsatisfactory to the user. A more prudent solution may be to appreciate that the bridge may well become available in the next turn and thus it is better not to move the units at all than to move them towards the second bridge. Thus a "prudent" cost function may not entertain paths which significantly detour from the more direct path.

5.0 Conclusion

Based on this discussion, it is possible to define a parameterized minpath algorithm that has diverse uses in a computer wargame. Since there are multiple needs for path finding and they have different needs, it is necessary to have a much generalized cost function which can be specialized based on the needs at any particular point in the program.

5.5 Bayes Belief Networks – Initial Investigation

John Rushing

1.0 Introduction

A Bayes belief network is a type of Bayesian learner. Bayesian learners are based on Bayes rule, and use probability estimates to perform classification and prediction. The primary distinction between Bayes belief networks and the commonly used Naïve Bayes classifier is that the belief networks model conditional dependencies between attributes, while the Naïve Bayes classifier makes the assumption that the attributes of interest are class conditionally independent of each other. In practice, many data sets contain attributes with conditional dependencies. For such data sets, Bayes belief networks may more accurately model the problem of interest than a Naïve Bayes classifier.

2.0 Initial Investigation

Bayes belief networks are described at a basic level in many data mining and machine learning texts such as (Han and Kamber), (Mitchell) and (Duda, ,Hart and Stork). A more detailed tutorial on the issues and techniques involved in construction and use of Bayes belief networks can be found in (Heckerman). What follows is a brief summary of some of the essential issues regarding Bayes belief networks.

Bayes belief networks consist of two components: a directed acyclic graph representing the conditional dependencies of the variables, and a set of conditional probability tables for each attribute. Determining the structure of the network from sample data is a difficult task, and in many cases domain knowledge of the user is used to give the network structure. In some cases, data are later used to refine the structure. There are some advanced algorithms to infer the structure directly. If there are no hidden variables and the network structure is known then computation of the conditional probability tables is straightforward, and proceeds in much the same way as the training of a traditional Bayes classifier.

One important thing to note is that most work with belief networks focuses on discrete attributes. Continuous valued attributes present significant additional complexities. The texts suggest that one way to deal with continuous attributes is to discretize them before using them with belief networks. This is commonly done for some other machine learning techniques such as decision trees or association rules. Commonly used discretization methods include simple binning, histogram based discretization, and entropy based discretization.

General inference in Bayesian belief networks has been shown to be NP-Hard. However, given a particular instance with all but one variable specified, it is possible to compute the probability for each possible value of the unspecified (output) variable from the conditional probability tables. This mode of operation corresponds to the typical usage of a supervised pattern classifier.

3.0 Development

The objective of the initial phase of development was to produce a basic implementation of the Bayes belief network and test it on some simple but realistic problems. Given this objective, and the information presented above, it was decided to limit the belief network in the following ways:

Pattern vectors must contain only discrete attributes
Dependencies between attributes must be specified a-priori
Inferences will be performed for only a single variable at a time

More advanced capabilities such as general inference, derivation of the network structure, and handling of continuous valued attributes may be added later as required. The belief network software will operate as a supervised pattern classifier. During the training phase, it will take a set of pattern vectors and a list of variable dependencies as inputs and produce a network including conditional probability tables as output. During the classification phase, the network will take a set of pattern vectors as input and produce labels for each of the pattern vectors.

3.1 Software Structure

The software delivered with this report compiles into two libraries (BayesNetworkLib and BayesDriverLib) and two executables (BayesNetworkTrain and BayesNetworkApply). The BayesNetworkLib library contains a basic implementation of the Bayes belief network, including training and classification methods, and methods to read and write representations of the network. Patterns are passed to the library as discrete valued (int) vectors. The BayesDriverLib library contains methods to read problem descriptions and variable dependencies in symbolic (string) form, convert them into discrete vectors, and call appropriate methods in the BayesNetworkLib to perform classification and training. The BayesNeworkTrain and BayesNetworkApply programs read parameters from a command line and use the BayesNetworkDriver class to perform network training and classification respectively.

Module	Description	Classes
BayesNetworkLib	Basic implementation of Bayes belief network	BayesNetwork, BayesNode
BayesDriverLib	Reads problem descriptions in symbolic form, drives the classifier in BayesNetworkLib	BayesNetworkDriver, PatternSet, Attribute
BayesNetworkTrain	Builds an executable to use BayesNetworkDriver to train a classifier	None
BayesNetworkApply	Builds an executable to use BayesNetworkDriver for classification.	None

3.2 Building the Software

A Microsoft Visual C++ 6.0 workspace file (BayesNetwork.dsw) and four associated project files (one for each library and executable) are located in the Source directory of the zip file. The libraries and executables can be built using these files. In addition, a CMakeList.txt file is provided so that with the use of the public domain CMake program, the source can be built on a wide variety of platforms.

3.3 Using the Executables

The BayesNetworkTrain and BayesNetworkApply programs are self-documenting. Running the programs with no arguments, incorrect arguments, or a –h option will prompt them to produce a usage message. The syntax for these executables is explained below:

```
Program: BayesNetworkTrain

Options:
  -b <filename>      Name of the Bayes Network file
  -h                 Print this message
  -i <filename>      Name of the input pattern file
  -p                 Print the network in readable form
  -s <filename>      Name of the structure file
```

```
  -t                    Perform consistency test
```

For BayesNetworkTrain the –b, –i, and –s options are all required, as they specify the two input files and the output file. The –p and –t options are useful for debugging and also provide a more human readable description of the network produced during training. The –p option prints out the conditional probability tables, and the –t option prints out an estimate of the likelihood of every combination of attributes. The –t option is not practical if the number of attributes is large, since the number of combinations is exponential.

```
Program: BayesNetworkApply

Options:
  -b <filename>       Name of the Bayes Network file
  -c <class>          Name of the class attribute
  -h                  Print this message
  -i <filename>       Name of the input pattern file
  -o <filename>       Name of the output pattern file
```

For BayesNetworkApply the –b, –c, –i, and –o options are all required, as they specify the names of the input and output files and the attribute that is to be predicted from the others. The output file will contain a data set in the same format as the input file, with the values of the specified attribute replaced by values estimated by the classifier.
The input data file must be a pattern set in ARFF format. The arff format has a simple header, which describes the attributes in the data set. The header is followed by a data block of pattern vectors, with one pattern per line of the file. Datasets must begin with a name declaration of the form:

```
@relation data_name
```

The name declaration must be followed by a series of attribute specifications of the form:

```
@attribute attribute_name {value_one, value_two, … value_N}
```

The attribute declarations are followed by the start of data tag:

```
@data
```

Following this tag is a list of pattern vectors, one vector per line. The elements of the pattern vectors should be separated by commas, spaces or tabs. The fish.arff file provided in the Fish subdirectory of the zip archive is an example of an arff file.

The structure file lists sets of variable dependencies, one dependency per line. The dependencies are of the form: var1, var2, where var2 depends on var1. The str.txt file provided in the Fish subdirectory is an example structure file.

The Fish subdirectory contains a sample problem derived from the Duda, Hart and Stork book. It consists of a data set with five attributes that describe the types and characteristics of fish caught at various places and times. One possible problem is to predict the type of fish based on the other parameters. The fish.arff file was divided at random into two sets: trn.arff and tst.arff, which can be used for training and testing the classifier. To create a classifier based on trn.arff and then classify the vectors in tst.arff, the following commands can be used:

```
BayesNetworkTrain -i trn.arff -s str.txt -b bayes.txt
BayesNetworkApply -i tst.arff -c fish -b bayes.txt -o res_tst.arff
```

The result is the file res_tst.arff, which can be compared to tst.arff to determine the accuracy of the classifier.

3.3 Using the Library Directly

The Bayes belief network library BayesNetworkLib can be used directly as a pattern classifier. The class BayesNetwork provides the methods that should be called to do network training and pattern classification. It also provides methods to read and write belief networks. The library expects discrete pattern vectors as input. The values for each attribute must be in the range (0 .. numValues-1). The BayesNetworkDriver class provides an example of how to call the library. The following examples demonstrate how to use the library to do training and classification:

Training a Bayes Belief Network

```
#include "BayesNetwork.h"

// Inputs to the training function, must be filled in
vector<BayesArc> arcs;
vector<int> dimensions;
vector<intvec> data;

// FILL IN VALUES HERE FOR arcs, dimensions, data

// Train the network
BayesNetwork network;
network.Train(dimensions, data, arcs);

// Save the network
network.WriteFile("bayes.txt");
```

Classifying Patterns Using a Bayes Belief Network

```
#include "BayesNetwork.h"

// Read the trained network
BayesNetwork network;
```

```
network.ReadFile("bayes.txt");

// Classify each pattern
for (int v = 0; v < numPatterns; v++)
  {
    vector<int> data;

    // FILL IN VALUES FOR data

    int clsId = network.Classify(data, clsAttr);

    // ClsId now has the result of the classification
  }
```

4.0 Initial Results

In order to test the software, a simple problem statement from the Duda, Hart and Stork book was used to generate input files for the classifier. The problem involves classification of fish based on the time and place they were caught and their physical characteristics. The problem has five attributes: *season, locale, fish, lightness*, and *thickness*. The type of *fish* caught depends on season and locale. The *lightness* and *thickness* both depend on the type of *fish*. The problem description also gives the legal values and conditional probabilities for each attribute. (These tables appear on p. 58 of the text. There is an error in the P(c|x) table, as the first row does not sum to one. I assumed a value of 0.2 for $P(c|x_3)$ to fix this.) Based on these tables, a program was written to generate data approximately matching these distributions. The program was used to generate a sample file with 100,000 pattern vectors (fish.arff). The structure file specifying the conditional dependencies is str.txt. Both files are in the Fish subdirectory. The fish.arff file was partitioned randomly into two parts. One part was used to train a Bayes belief network, and the other to evaluate its accuracy. In addition, the same data files were used to train and test a decision tree classifier, namely the C4.5 implementation in the Orange data mining toolkit. The Python scripts used to perform the processing, and the results are included in the Fish subdirectory of the zip archive. The Bayes network training software printed out the following conditional probability table, which was verified to be correct:

Conditional Probability Tables for the Fish Sample Problem

```
Attribute season:
P(season = Winter | ) = 0.250660 (0)
P(season = Spring | ) = 0.251400 (1)
P(season = Summer | ) = 0.246840 (2)
P(season = Autumn | ) = 0.251100 (3)
Attribute locale:
P(locale = NorthAtlantic | ) = 0.599130 (0)
P(locale = SouthAtlantic | ) = 0.400870 (1)
Attribute lightness:
```

```
P(lightness = Light_ |  fish = Salmon_) = 0.599365 (0)
P(lightness = Medium |  fish = Salmon_) = 0.202687 (1)
P(lightness = Dark__ |  fish = Salmon_) = 0.197948 (2)
P(lightness = Light_ |  fish = SeaBass) = 0.196034 (3)
P(lightness = Medium |  fish = SeaBass) = 0.299271 (4)
P(lightness = Dark__ |  fish = SeaBass) = 0.504695 (5)
Attribute thickness:
P(thickness = Wide |  fish = Salmon_) = 0.300387 (0)
P(thickness = Thin |  fish = Salmon_) = 0.699613 (1)
P(thickness = Wide |  fish = SeaBass) = 0.601129 (2)
P(thickness = Thin |  fish = SeaBass) = 0.398871 (3)
Attribute fish:
P(fish = Salmon_ | season = Winter locale = NorthAtlantic) = 0.500668 (0)

P(fish = SeaBass |  season = Winter locale = NorthAtlantic)
= 0.499332 (8)
P(fish = Salmon_ |  season = Spring locale = NorthAtlantic)
= 0.601002 (1)
P(fish = SeaBass |  season = Spring locale = NorthAtlantic)
= 0.398998 (9)
P(fish = Salmon_ |  season = Summer locale = NorthAtlantic)
= 0.403883 (2)
P(fish = SeaBass |  season = Summer locale = NorthAtlantic)
= 0.596117 (10)
P(fish = Salmon_ |  season = Autumn locale = NorthAtlantic)
= 0.201054 (3)
P(fish = SeaBass |  season = Autumn locale = NorthAtlantic)
= 0.798946 (11)
P(fish = Salmon_ |  season = Winter locale = SouthAtlantic)
= 0.699485 (4)
P(fish = SeaBass |  season = Winter locale = SouthAtlantic)
= 0.300515 (12)
P(fish = Salmon_ |  season = Spring locale = SouthAtlantic)
= 0.807966 (5)
P(fish = SeaBass |  season = Spring locale = SouthAtlantic)
= 0.192034 (13)
P(fish = Salmon_ |  season = Summer locale = SouthAtlantic)
= 0.102626 (6)
P(fish = SeaBass |  season = Summer locale = SouthAtlantic)
= 0.897374 (14)
P(fish = Salmon_ |  season = Autumn locale = SouthAtlantic)
= 0.296223 (7)
P(fish = SeaBass |  season = Autumn locale = SouthAtlantic)
= 0.703777 (15)
```

The Bayes belief network and decision tree produced very similar results. The confusion matrices for each of the classifiers (generated by the ADaM ITSC_Accuracy tool) are

shown in the following figure. The classifiers produced similar quality of results, with accuracies within half a percent of each other. This is not surprising given that the decision tree is actually using very similar information to perform its classification given that there only four predictive attributes.

Results for Bayes Belief Network	Results for Orange C4.5 Classifier
Classes 2, Samples 50000 Confusion Matrix | 0 1 <--- Actual Class ----------	Classes 2, Samples 50000 Confusion Matrix | 0 1 <--- Actual Class ----------
0 | 16889 5737 1 | 5633 21741 ^ | +------ Classified As	0 | 15737 4614 1 | 6785 22864 ^ | +------ Classified As
POD 0.791215 FAR 0.205779 CSI 0.656610 HSS 0.540879	POD 0.832084 FAR 0.228844 CSI 0.667309 HSS 0.535478
Accuracy 38630 of 50000 (77.260000 %)	Accuracy 38601 of 50000 (77.202000 %)

5.0 Future Experiments

The next step is to apply the belief network to a problem related to ground based wargames. Some potential problems of interest include detection of weak points in front lines, and detection of danger conditions for specific units. (For example, determining if a unit is in danger of being surrounded, overrun, losing its supply line etc.). These problems will be simulated by generating realistic input data representing the situations. The belief network will be evaluated on this data, and compared to other methods.

6.0 References

R. Duda, P. Hart, D. Stork, *Pattern Classification*, Wiley and Sons, 2001.

J. Han, M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2001.

D. Heckerman, "A Tutorial on Learning with Bayesian Networks", Microsoft Technical Report MSR-TR-95-06, 1995.

T. Mitchell, *Machine Learning*, McGraw Hill, 1997.

5.6 Embedding AI Technologies in General Applications

John Tiller

1.0 Introduction

This report address the technical issues associated with structuring AI technologies and their implementation so that the resulting code is directly callable from general applications. The result of this study has particular application to the implementation of AI technologies in computer wargames, where it is necessary to embed AI techniques into the computer game implementation without releasing control of the user interface or processing to the technology implementation.

2.0 Overview

There are four specific technologies that are to be considered in this report: Neural Networks, Expert Systems, Genetic Algorithms, and Bayesian Networks. With respect to a general AI technology there are 3 to 5 general steps associated with interacting with the AI code from a general application:

- Populate the algorithm with specific data, in the format of the algorithm, or in object-oriented terms, invoke an object constructor.
- Optionally or in conjunction with the first step, invoke initialization code associated with the technology.
- Invoke the specific routine associated with the AI technology processing.
- Possibly inquire into specific data or results associated with this processing.

The specific routine associated with the AI technology is assumed to either return or generate data or results as a consequence of its processing, or alternatively, to invoke other processing in the AI engine through callbacks.

3.0 Specific Implementations

This report now looks at four specific AI technologies and the specific requirements of each implementation.

3.1 Neural Networks

The specific implementation of neural networks considered here is called **Think**. It is based on an implementation of neural networks by the author written in C++. The particular sequence of calls involved depends on the specific neural network.

3.1.1 Back-Propagation

The implementation of Back-Propagation involves three steps:

- Declaration of the object constructor.
- A call to **Train** based on the training data.

- For each instance of the input data, a call to **Recall** to produce the corresponding output data.

3.1.2 Counterpropagation

The implementation of Counterpropagation involves three steps:

- Declaration of the object constructor.
- A call to **Train** based on the training data.
- For each instance of the input data, a call to **Recall** to produce the corresponding output data.

3.1.3 Hopfield Networks

The implementation of Hopfield Networks involves three steps:

- Declaration of the object constructor.
- For each training pattern, a call to **Encode.**
- For each input pattern, a call to **Recall** to produce the corresponding output pattern.

3.2 Expert Systems

The specific implementation of expert systems considered here is called **Reason.** It is based on an implementation of expert systems by the author written in C++. The invocation of the expert system involves five steps:

- Declaration of the object constructor.
- For each rule, a call to **Learn** so the rule is stored.
- For each fact, a call to **Assert** so the fact is stored.
- A single call to **Reason** to invoke the expert system.
- Inquiry of the output fact from the call, or inquiry of the validity of other facts in the resulting logic database.

3.3 Genetic Algorithms

The specific implementation of genetic algorithms considered here is called **Evolve.** It is based on an implementation of genetic algorithms by the author written in C++. The invocation of the expert system involves three steps:

- Declaration of a population object constructor.
- A call to **Evolve** based on a particular objective function.
- Inquiry of the resulting population.

3.4 Bayesian Networks

The specific implementation of Bayesian networks considered here is called **Decide.** It is based on work done by John Rushing at the University of Huntsville-Alabama and is written in C++. The invocation of the Bayesian network involves three steps:

- Declaration of the object constructor.
- A call to **Train** based on input data, possibly followed by saving the result.

- For each input pattern, a call to **Classify** to classify the input pattern.

4.0 Conclusion

With the right software design and approach, the implementation of AI technologies in general applications can be efficiently achieved if the right object-oriented classes are identified and the specific steps required to implement the technology defined as the appropriate member functions. This allows applications such as computer wargames to invoke the AI technology from within its particular AI engine and to make use of the results again within the engine as opposed to the context of some higher level-driver or higher-level interface.

5.7 Applications of Expert Systems to Computer Wargame AI Engines

John Tiller

1.0 Introduction

Expert Systems are a way of describing logical structures containing a number of entities of the two types:

- Facts – data which is known to be true.
- Rules – logical inferences based on known facts with logical conclusions.

In general Expert Systems are non-procedural. That is, they execute based on the facts and rules they contain but in no particular order. The specific execution order is determined by the *assertion* of new facts, which causes rules to be activated based on this new knowledge.

2.0 Specific Implementation

The author has developed one such Expert System in C++ called **Reason**. This Expert System has an internal engine as well as a high-level parser so that it can be used both as a stand-alone application as well as being able to be embedded in other applications and callable as subroutine code.

The purpose of this report is to examine the application of such an Expert System to two instances of computer wargames: one having to do with a ground game and the other having to do with an air power game.

3.0 Ground Game Example

In Figure 1 is an example of a Ground Recon Expert System written in the Reason language. Each Expert System starts with a number of rules based on an *initial fact*. This initial fact is always true and thus rules which are based on it are always invoked. The purpose here is to establish starting conditions for the logic, namely that we are starting under conditions of Event 1 and we are establishing a safety distance of 6 units.

Having established those facts, the logic then proceeds to the advancing stage. A random location on the given map is determined and an order is given to the recon element to advance to that location in search of the enemy.

Once the recon element finds the enemy and is determined to have advanced closer than the safety distance, the logic proceeds to Event 2 and Event 1 is retracted so that it is no longer valid.

Establishing Event 2 causes the main body of the force to advance to meet the enemy. Meanwhile the recon element retires until it is a safe distance from the enemy.

In this way, the natural logic that ground forces would follow is coded in the non-procedural language of the Expert System and the processing occurs as necessary based on the circumstances.

```
// c l a s h . e x p
//
// Clash expert.

// Start with Event 1.
(initial_fact) => assert (Event 1)
(initial_fact) => assert (Safety 6)

// Recon units should search out the enemy.
(Event 1), (Recon #org), !(Objective #org #x), (Width #w),
    (Height #h) => assert (Advance #org ((? #w) (? #h)))

// Once the recon makes contact with enemy, transition to Event 2.
(Event 1), (Recon #org), (Distance #org #d), (Safety #s), (< #d #s),
    (Location #org #x)  => assert (Event 2), retract (Event 1),
    assert (Enemy #x)

// While seeking out the enemy, main body should advance slowly.
(Event 1), (Org #org), (Main #org) => assert (Advance #org slow)

// Once the enemy is engaged, main body should advance to make
contact.
(Event 2), (Org #org), (Main #org), (Enemy #x) =>
    assert (AdvanceTo #org #x)

// While at a distance from enemy, recon should advance.
(Recon #org), (Distance #org #d), (Safety #s), (> #d #s) =>
    assert (Advance #org slow)

// Once engaged with enemy, recon should fall back.
(Recon #org), (Distance #org #d), (Safety #s), (<= #d #s) =>
    assert (Fallback #org)
```

Figure 1. Example of Recon Expert System

4.0 Air Power Game Example

In Figure 2 is an example of logic associated with a SAM Site Expert System. The basis for the logic is the assumption that SAM Sites will be either *dormant* or *non-dormant*. If dormant, then the site may or may not become active, but in any event, will not launch missiles. A non-dormant SAM Site will launch missiles when the opportunity arises.

The determination of what SAM Sites are non-dormant is determined to a certain extent to be random, with the exception that "overlapping" SAM Sites will be both dormant or both non-dormant. This is to present a consistent behavior for the purpose of IADS.

In the Expert System, basic data concerning the range of the SAM Sites is established. It is also assumed that additional data such as the distance between SAM Sites is known. Finally, it is assumed that there are one or more enemy flights in the area and that the range to those flights is known.

The Expert System begins by making a random determination of which sites will be non-dormant. It then determines which sites are overlapping and ensures that the assignment of dormant or non-dormant is consistent for all of them connected in this way.

Finally, if a legitimate target flight appears and the site is known to the non-dormant, then a command to fire a missile at the target flight is asserted.

This approach uses the available data in a non-procedural way that avoids having to do various looping constructs in a high level programming language. This allows the logic to be separated from the programming of the logic and allows it to stand alone where it can be viewed and easily modified.

```
// s a m _ s i t e . e x p
//
// Expert system performing SAM site logic.

// Define test data.
(initial_fact) => assert (sam_site S1), assert (range 50 S1)
(initial_fact) => assert (sam_site S2), assert (range 50 S2)

// Randomly assign non-dormant sam sites.
(sam_site #X), (<= (?) 0.1) => assert (non-dormant #X)

// Two sam sites with one in range of the other are in the same
component.
(sam_site #X), (sam_site #Y), (distance #D #X #Y), (range #R #X),
    (<= #D #R) => assert (component #X #Y)

// When two sam sites are in the same component and one is non-
dormant, then
// the other is non-dormant.
(sam_site #X), (sam_site #Y), (component #X #Y), (non-dormant #X) =>
    assert (non-dormant #Y)

// If a non-dormant sam site has a target, then it can fire.
(sam_site #X), (flight #F), (can_fire #X #F), (non-dormant #X) =>
    conclude (fire #X #F)
```

Figure 2. SAM Site Expert System

5.0 Conclusion

Using an Expert System to handle AI logic within a wargame has several advantages:

- The Expert System high-level code serves as a way of documenting the internal logic of the AI separate from the program itself.
- The Expert System high-level code provides the end-user with a way of modifying the AI logic without having to have access to or be able to program in the programming language of the application.
- An approach using an Expert System introduces non-procedural processing into the AI engine which may provide more intricate logical processing than would occur with strict procedural code, although this can have both positive and negative impact.

The language of the Expert system provides a way of naturally describing the logic associated with the AI engine without the details of a programming language such as C++. The resulting statements can be read and understood by themselves without having to understand the execution context they will occur in. This then results in a very flexible, transparent, and modifiable AI game engine for computer wargames.

5.8 AI Agents within an HLA Federation

John Tiller

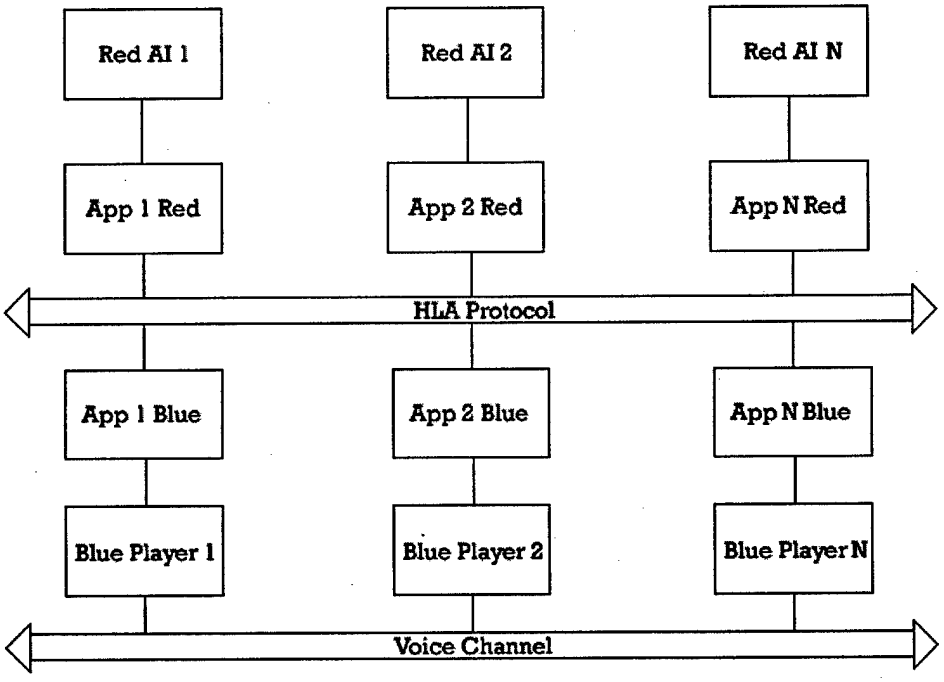
1.0 Introduction

The report looks at the issues associated with incorporating AI Agents within a High Level Architecture (HLA) federation of simulations. The HLA standard was developed by the Department of Defense and designed to be used as a standard means for multi-simulation combinations, or federations, to be built so that the individual simulations can communication their respective state information to each other and thus result in a combined simulation environment. One interesting application of this technology is in the development of the DARPA DARWARS joint training project, which is intended to support ad-hoc joint training exercises over the Internet using training simulations specific to the various branches of the military and their respective disciplines. For example, a joint training exercise could be created dynamically using an Army ground-based simulation, a Navy simulation, a Marine simulation, as well as having a pilot in a flight simulator and a forward-observer using an artillery simulation to call fire support. The HLA protocol is responsible for communicating state information from one application to the others so that a consistent joint world-view is generated for all players in the exercise. In a joint training exercise such as this, the players would then communicate to each other using a Voice-Over-IP application so that they could communicate commands and information to each other in their respective sides.

It would not be uncommon in such an ad-hoc training exercise, or even a scheduled exercise, to have a need for AI Agents to play the role of one or more players in this federation. Indeed, since the intention is to train our military forces in our doctrine, it would be best if all of the human players could participate on the friendly side and the enemy side was under AI control, thus not requiring that any human players opt out of

the friendly-side training and also providing perhaps a more challenging opponent for the human players. Likewise, if one or more human players are not available to server all of the necessary roles in a particular joint training scenario, then it would be advantageous to have AI Agents to play these roles and thus enable the exercise to take place at all.

The figure below illustrates one such possible federation where AI Agents are playing the role of the enemy commanders while human players are playing roles as the friendly commanders. Notice that the human players have a separate voice channel that they are using to communicate with. The needs and requirements of the AI Agents that are necessary to complete this diagram into a working design are investigated in this report.



2.0 Issues

In the design of how AI Agents could be incorporated into an HLA federation, there are several issues that must be addressed:

- What do the AI Agents communicate to each other?
- How do they communicate this information, within the HLA protocol or is a separate communication channel required?
- When there is a mixed side consisting of both human and AI players, how is the communication between human and AI players handled?

The answer to the first question depends on the structure of the AI Agents. For the purpose of this report it will be assumed that the AI Agents are programmed to follow logic based on an Evaluate-Plan-Execute (EPE) loop. That is:

- **Evaluate** – Evaluate the current situation both with respect to friendly and enemy forces.
- **Plan** – Based on that evaluation, create plans on course of action.
- **Execute** – Using these plans, execute them and resolve the action resulting from them.

Based on this assumption, then each part of the EPE loop would need to be communicated to the other agents on the same side:

- What is the evaluation of the agent's forces and opposing forces? In military terms, this is called a situation report.
- What plans does each agent have and how do these plans relate to each other?
- How is the execution of these plans coordinated between the agents?

Since coordination between agents is necessary, it would like be the case that the optimal approach would involve designating one of the agents as being the so-called Joint Commander, having overall command of a particular side. Indeed, in human-based joint training exercises, this would likely be the case as well so that individual actions by the players would have a basis for coordination.

Using a Joint Commander approach, then one of the AI Agents would be responsible for creating an overall plan based on the information received by that agent from the others. This overall plan would then be communicated back to the individual agents who would be responsible for generating lower-level plans from it. In military terms, these lower-level plans are called a frag (or fragmentation) order. Finally, the Joint Commander would then be responsible to coordinating the individual execution of the plan so that related tasks such as conducting air support attacks before advancing on the ground was accomplished.

3.0 AI Agent Communication

After the issue of what the AI Agents communicate and how this communication is used and coordinated is addressed, the remaining question is whether this information can be conveyed within the HLA protocol or whether a separate AI communication channel must be established, much like human players would establish a separate voice channel for their high-level communication.

HLA is based on an object-model with single inheritance. A Run-Time-Interface (RTI) exists to communicate information between applications in a federation. Within the context of a federation, it is possible for applications, or federates, to declare Object Classes, declare Object Instances, and to communicate the value of Object Attributes for an object instance to other members of the federation. Because the purpose of the HLA is to communicate these Object Attribute values to other members of the federation, the HLA Object Model does not extent to member functions and all attributes of the objects are readable.

In addition to standard object classes, HLA supports the definition of Interaction Classes. Interaction Classes represent actions which may be taken by one federate which could impact other federates. This feature is used to allow objects from different federates to interact in the joint simulation.

Based on this design, then a possible way for AI Agents to proceed in a federation would be as follows:

- The AI Agents publicize their object models of evaluation, planning, and execution to the other agents in the federation.
- Based on a prior design protocol, the agents could negotiate for the role of Joint Commander.
- The Joint Commander agent could then poll the AI Agents for evaluation data, coordinate the generation of planning data, and then coordinate the execution of planning data between the agents using the object classes previously defined.

4.0 AI Agent to Human Communication

When the AI Agents are used as participants in an otherwise human team, then there is a need to have communication between those agents and the human player in ways that work with both the processing of the agents and the communication levels of the human player. In one direction, from agent to human player, it would be possible to incorporate Text-to-Speech technology in the AI agent, or in the communication infrastructure of the human player, so that information and commands generated by the AI Agent could be verbalized to the human player. In this way, it would naturally be incorporated into the default voice-based communication of the other players.

In the other direction, from human player to AI Agent, two possible approaches are possible. In one approach, speech recognition could be incorporated so that the voice commands and voice information of the human players could be converted into data that could be recognized by the AI Agent. A simple protocol for communicating in this way would need to be designed so that the resulting information was simple and unambiguous to the AI Agent, given the current limitations in AI language processing. Alternatively, the human player using the application interface of their simulation, could enter these commands and information using standard mouse-and-keyboard commands. While this would be less efficient than the voice recognition approach, it may work better depending on the specific application environment and players involved. In any event, it would probably be prudent to incorporate this ability as a back-up to speech recognition even if that was the preferred approach.

5.0 Conclusion

In HLA federations, especially those being used for joint training, the need for AI Agents to act as players for various sides and roles is clear. The implementation of these AI Agents can be done in such a way that a coordinated course of action results, whether these agents are acting by themselves on one side or in mixed teams with human players.

5.9 Implementing National Characteristics in Wargame AI

John Tiller

1.0 Introduction

The development of a challenging and adaptable AI for computer wargames also involves providing the human player with a programmed opponent that reflects certain national characteristics. Doing this provides the human player with training and experience in potential enemy responses and possible ways to respond to those characteristics. At the very least, having these characteristic responses in the AI means that the human player has been exposed to them and they are not completely unknown to the player. This report looks at how such national characteristics can be programmed into wargame AI.

2.0 Overview

The author has developed over 30 commercial computer wargames and has encountered the issue of national characteristics in several instances. Two good examples of these are with respect to the Japanese and Russian armies during World War II:

- The Japanese soldier was taught that surrender was a dishonorable action, regardless of the situation. Likewise, the soldier understood that the suicide charge, or Banzai Charge, was an honorable way to respond to a hopeless situation.
- The Russian army felt that maneuvering around enemy minefields was a military mistake. Their experience had been that when this was done, it only exposed their forces to devastating fire purposely sited at the bottlenecks that resulted. Although contrary to the Western way of thinking, they felt that it was actually more humane to send their forces directly through enemy minefields and take the resulting casualties, than to expose their forces to devastating enemy fire and incur much higher casualties.

With respect to the non-Russian Allied forces in World War II, there was also a sensitivity to casualties. Particularly in the case of the British, there was the issue that high casualties, even in the case of a significant military victory, were not politically acceptable to the nation and had to be avoided in order to retain popular support for the war at home.

3.0 Implementation

There are a number of ways that national characteristics can be implemented in a computer wargame so that the resulting effect both induces the correct behavior from the AI and also limits the human player to respond in ways that reflect national characteristics:

- **Parameterization** – It is possible to introduce numeric parameters which can be calibrated in value to attain the proper effect.

- **Rule Effects** – It is possible to program additional rules that reflect national characteristics and then have these rules only apply to the specific nationalities involved.
- **Victory Conditions** – It is possible to introduce modifiers to the default victory conditions in the wargame so that effects that transcend the military outcome can be incorporated into the final victory determination.

These techniques are considered in detail in the following.

3.1 Parameterization

In the case of the Russian behavior towards avoid minefields, the AI can be calibrated so that the corresponding effect is achieved through the use of parameters in the minpath algorithm. The AI uses the minpath algorithm to determine paths for the units under AI control. This algorithm uses the standard movement costs associated with various terrain in the scenario as well as additional effects such as how disruption of the units involved can increase their movement costs. In the case of this national characteristic, it is possible to introduce a *minefield parameter* which reflects the "cost" of moving units through known minefields. This parameter value can be set depending on the effect required:

- When the minefield parameter is set high, it indicates that there is a high cost associated with moving through minefields. This will cause the minpath algorithm to avoid minefields except in extreme situations where no other path is available.
- When the minefield parameter is set low, then the minpath algorithm will readily find paths that include minefields. This would then reflect the Russian national characteristic to not avoid minefields.
- The minefield parameter can also be set so that movement through minefields is considered impossible and thus the minpath algorithm would always find an alternative path, or refuse to move the units at all if no alternative were available. In many cases, this could reflect more Western notions of maneuver.

3.2 Rule Effects

In the case of the Japanese Banzai Charge, a new set of rules needs to be introduced which implements this effect in the wargame. Further, the game engine is coded so that it recognizes that the new rules only apply to Japanese forces. These new rules would then implement the characteristics of the Banzai Charge and the extent to which they could be utilized by the Japanese side. In this way, the new rules not only induce the proper AI behavior in the Japanese AI, but also support this national characteristic in the case of a human player on the Japanese side.

In addition, the extreme reluctance to surrender can also be programmed with an additional rule, which precludes surrender for Japanese units. Again, this results in the correct behavior for the Japanese side, in both the case of the AI player and the human player, since it is made part of the basic rules of the game engine.

3.3 Victory Conditions

In the paper, "Casualty Budgets - with applications to both military planning and commercial wargaming", Col Matthew Caffrey, USAFR, and the author of this report, developed the concept of a *Casualty Budget* for a particular participant in military conflict. This notion is based on the principle that in certain situations, a participant in military conflict is constrained by the number of casualties they can suffer and still retain political support for the conflict. This has been seen in many historical instances such as the Western Allies in World War II and the American Army in the Vietnam War. When the casualties for an affected side reach a certain level, there is an impact to the victory conditions for the side that transcend the pure military situation. This affects the ability of the commander for that side to pursue military solutions and to take into account the political impact of their decisions.

In a computer wargame, there is always a calculation of victory associated with outcome of the scenarios. In most every case, this calculation takes into account casualties for both or at least one side. The implementation of a Casualty Budget then modifies that calculation by imposing an upper bound on the allowed casualties for a particular side. When this upper bound is invoked, it then forces the victory calculation towards a loss for the affected side that is independent of the casualties for the opposing side or the objective points that the affected side might acquire.

The net result of this effect is to compel the affected commander to reduce the tempo or intensity of their attack or fighting in general. This effect would apply to both an AI opponent, programmed to account for the effect, and a human player who was playing in an optimal manner.

4.0 Conclusion

This report has addressed how there are multiple ways in which national characteristics can be implemented in a computer wargame in such a way that the desired behavior is generated in an AI opponent and also how national characteristics for both the AI and human player can be generated. These implementations range from simple numeric parameterization of certain effects to programmed rule changes, and additional rule effects specifically designed to induce characteristic behavior from the player, be it AI or human.

5.10 The Technical Implementation of Plug-and-Play AI in a Computer Wargame

John Tiller

1.0 Introduction

It is desirable in the design of an adaptable computer AI in computer wargames to have a design that allows for new technologies to be introduced in a plug-and-play fashion. That is, by having a flexible design that allows for new technologies to be introduced without requiring access to or recompilation of the wargame source code. With this achieved, it

is possible for third-parties or even the end-users themselves to make changes to the AI implementation and thus have a high degree of flexibility.

2.0 Implementation

In most computer operating systems, the concept of dynamically invoked code is implemented through dynamically-linked libraries. Such libraries exist as separate files from the main executable, but are dynamically-linked at run-time with the main program code to form the final run-time executable. In this way, new versions of the library code can be introduced without requiring recompilation of the main program code or access to its source code. This report considers this approach under the Microsoft Windows operating system.

3.0 Technical Issues

There are two technical issues regarding the implementation of Plug-and-Play AI in this design. The first technical issue is how to code and compile a Windows dynamically-linked library, or so-called DLL file. To achieve this, the functions in the DLL must be declared as export functions. This is done by using the Win32 declaration:

```
__declspec(dllexport)
```

To facilitate this in code development, a declaration is typically introduced:

```
#define dllfunction __declspec(dllexport)
```

This then results in function declarations for functions in the DLL having the general form:

```
dllfunction void __stdcall function ();
```

The Microsoft Visual C++ compiler can then be configured to compile this code as a DLL file.

The next technical issue has to do with how to structure the DLL code so that it interfaces with the other code in the game engine. This assumes that the game code is designed using the Model-View-Controller design pattern with the game engine encapsulated in the Model section of the design. Here we will assume that the top-level class of the Model code is declared to be `Model`. This class is assumed to have certain public member functions which describe the actions and behavior of the game engine itself. In a standard in-line design of the AI code, the AI functions would then call these member functions to determine the state of the game engine and the impact of game rules.

In the design of these Plug-and-Play AI components, it is necessary to design them in a more third-party manner. However, it is necessary that these external AI calls must still have access to game state and rule information. For this reason, it will be the case that

most every call to the external code will pass a pointer to class `Model` as the first parameter. In a sense, this represents the abstracting of the internal calls using the so-called `this` pointer. With this understanding, we can then understand that most calls to the DLL code will have the general declaration:

```
Dllfunction void __stdcall function (const Model
*model, ...);
```

Notice that the pointer to class `Model` is declared to be `const` meaning that the external AI code will have access to the game state and rule information, but not be able to change it directly.

4.0 Conclusion

This report has considered the technical details involved in coding and compiling external Plug-and-Play AI routines for use in computer wargames. This turns out to be a straight-forward implementation of the Windows DLL concept. The resulting implementation has the advantage that the external code can then depend or be implemented in a wide-variety of AI techniques without requiring a redesign of the AI engine or even its recompilation.

5.11 DBSCAN for Determination of Front Lines

John Rushing

1.0 Introduction

There are many interesting problems in ground-based wargames related to identification of front lines. The AI for a wargame should be able to determine where the fronts are, which units constitute the front, and the locations of any gaps or weak points in the front. There are many possible ways to address these problems, including the use of general-purpose segmentation and clustering algorithms. This report demonstrates how one such technique, Density Based Spatial Clustering of Applications with Noise (DBSCAN), can be used to identify groups of units that constitute front lines.

2.0 DBSCAN Algorithm

DBSCAN is a density based clustering algorithm. It identifies groups of samples or patterns that are connected by contiguous dense regions in the pattern space. Unlike many commonly used clustering techniques, it is capable of forming clusters of arbitrary shape and size. The algorithm scales very well as it only requires a single pass through the input data set. DBSCAN is capable of identifying outliers, or patterns that do not fit well in any particular cluster. DBSCAN is typically used in clustering applications with sets of continuous variables. These variables constitute an N dimensional space where each dimension or axis corresponds to a variable. The variables are often normalized to have

the same range so that they are given equal weights by the distance function. There are several definitions, which are key to the algorithm (Ester, 1996):

- **Epsilon Neighborhood:** The epsilon neighborhood of a pattern or objects is the area within a radius epsilon of the object. Epsilon is a parameter for DBSCAN.
- **Core Object:** An object is a core object if its epsilon neighborhood contains at least minPoints objects. MinPoints is an input parameter for DBSCAN.
- **Directly Density Reachable:** An object p is directly density reachable from object q if p is within the epsilon neighborhood of q and q is a core object.
- **Density Reachable:** An object p is density reachable from object q if there is a chain of objects p1, p2, ... pn, p1 = q pn = p such that pi+1 is directly density reachable from pi
- **Density Connected:** Objects p and q are density connected if there is an object o such that both p and q are density reachable from object o

A density-based cluster is a set of density-connected objects that is maximal with respect to density reachability. In other words, it is the largest set of objects that can be formed from a particular core object such that all objects in the cluster are density reachable from each other. Any object not in a cluster is considered an outlier (noise). DBSCAN first finds all core objects, and then iteratively adds all objects reachable from each core, merging cores as necessary. The algorithm is O(n log n) using a spatial index, $O(n^2)$ otherwise. The performance depends on the size of the neighborhood and whether there is an efficient way to identify neighboring objects in the pattern data. The input parameters epsilon and minPoints have significant impact on the performance of the algorithm. The settings for these parameters vary by problem domain and data set.

3.0 Application of DBSCAN for Ground Based Wargames

DBSCAN can be used to identify fronts in ground-based wargames by identifying continuous groups of units that are close enough together to provide mutual support. Friendly units that constitute a continuous front line will be grouped into a single cluster, while isolated units will be identified as outliers. Gaps in the front will be exposed, as the units on either side of the gap will be in different clusters. The epsilon and minPoints parameters are used to define the conditions for a viable front line. The algorithm can be applied multiple times with varying levels of epsilon and minPoints to identify lines of varying strength.

The DBSCAN algorithm assumes that there is a meaningful distance measure that is defined for the input patterns. In the case of an N-dimensional pattern space, distance measures such as Manhattan distance and Euclidean distance are commonly used. For ground-based wargames, it makes sense to consider the movement cost of the shortest viable path from unit to unit. This path must take into account variable movement costs for different types of terrain, impassable hexes, features such as roads, rivers and railroads, and the presence of enemy units. It may be desirable to vary the radius by unit type, as some units are much more mobile than others.

Another important consideration is the strength of the units. A group of very weak units may not constitute a viable front, even if there are large numbers of them. The total strength of the units in a particular region is therefore more interesting than the number of units. The DBSCAN algorithm can be modified to accomplish this simply by summing the strength of the units in the epsilon radius. In this case, the minPoints parameter becomes a minStrength parameter instead. Assignment of a strength rating will depend on the game engine and rules of the game, and may consider factors such as the size of the unit, its supply state, its morale level, any attrition it has suffered and other factors.

4.0 Development

The objective of the initial phase of development was to produce a basic implementation of DBSCAN algorithm, adapt it for use in ground-based wargames, and test it on some simple but realistic problems. In order to do this, a simple wargame model and visualization module were required. The goal of these modules was to allow some scenarios to be formulated to test the operation of the DBSCAN algorithm on some simplified data. For future phases of development, a more comprehensive and realistic game model could be used instead. The game model provides important information to the DBSCAN algorithm, such as unit to unit distances and unit strengths.

4.1 Software Structure

The software delivered with this report compiles into a library and an executable. The library contains a basic wargame model, and the executable contains the DBSCAN algorithm and a small command line driver for the algorithm. There is also a directory with a set of icons used by the program.

Module	Description	Classes / Functions
Model	A very simple wargame model, including a data reader and viewer	Unit, Model, Viewer, Reader, Image, Point
DBSCAN	An implementation of DBSCAN built on top of the wargame model	DBSCAN Algorithm, Driver Program
Icons	A set of icons (images specified in text files) used by the program	Icons, *.txt

4.2 Building the Software

A Microsoft Visual C++ 6.0 workspace file (DBSCAN.dsw) and two associated project files (one each for the library and executable) are located in the Source directory of the zip file. The libraries and executables can be built using these files. In addition, a CMakeList.txt file is provided so that with the use of the public domain CMake program, the source can be built on a wide variety of platforms.

4.3 Using the Executable

The DBSCAN program is self-documenting. Running the program with no arguments, incorrect arguments, or a –h option will prompt them to produce a usage message. The syntax for this executable is explained below:

```
Program: DBSCAN

Options:
  -h                        Print this message
  -e <epsilon>              Epsilon radius for DBSCAN
  -i <filename>             Name of the input model
  -o <filename>             Name of the output image
  -s <strength>             Min strength for core objects

Description:
  The DBSCAN program reads a game model (including map
  and units) and plots it to an image file.
```

The –i, and –o options are required as they specify the input and output file. The –e and –s options specify the conditions required for core objects, which controls how strict the conditions on the clusters are. The input data set is a wargame model, specified in two ASCII text files: one for the units and another for the map. The model file contains a size in hexes and a reference to the map file, and then a list of units. The units are specified one per line. Each line has the following information, delimited by the "-" character: unique id, nation, unit type, attack strength, defense strength, movement allowance, starting hex, and flag word. There are two legal values for nation (Red and Blue) and three types of units (Inf, Cav, Art). The map is very simple and is assumed to consist of four types of terrain: clear, rough, mountain, and ocean. The map file has a size in hexes as the first line, then lists of hexes for rough terrain, mountains and oceans. Hexes are assumed to be clear if no terrain modifier is specified. The output of the executable is a binary image file.

Several sample model files are provided in the top level directory: model*.txt. They all reference the same map file, Map.txt. A sample Python script Plot.py runs DBSCAN on the model files and produces gif images from them. The script uses the ITSC_CvtImageToGif program and the file img.ctbl to convert the binary images to gifs. Sample results are included in the top level directory and are referenced later in this report.

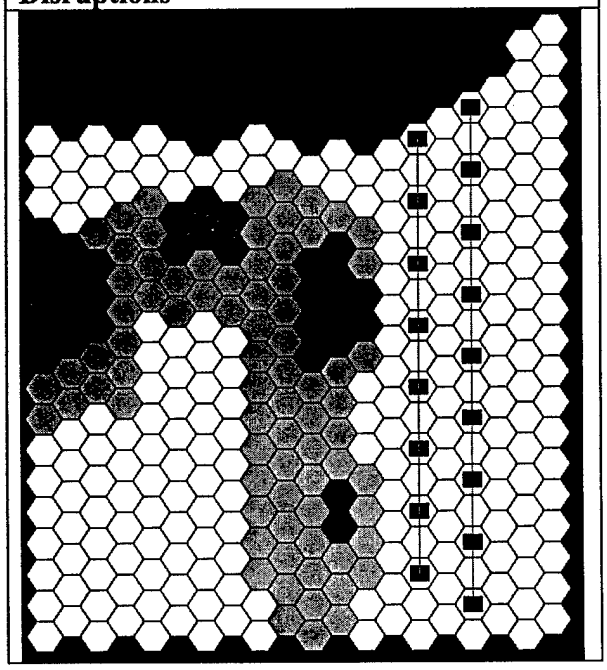
5.0 Sample Results

In order to test the software, some simple cases were devised. In each of the figures in this section, there are two opposing sides, Red and Blue. There are infantry units with defense strength of 5 and cavalry units with defense strength of 2, represented by the standard symbols. There are four types of terrain: clear, rough, mountain, and ocean. The clear hexes are white, rough hexes tan, mountain hexes brown and ocean hexes blue. The clear hexes have a movement cost of 1, rough hexes 2, and the mountains and ocean are assumed to be impassible.

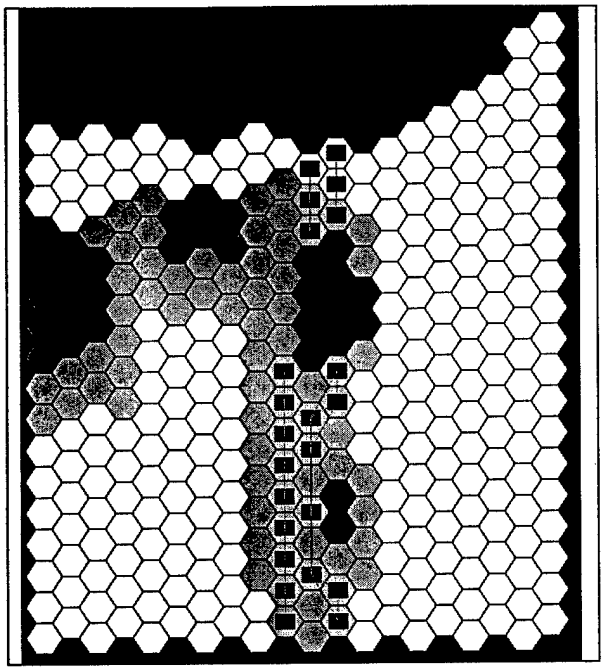
There are five cases illustrated below. The units forming the line are linked by green lines plotted from center to center of the connected units. A cluster consists of all connected units. Units connected by plotted lines are within each others epsilon radius. The first scenario is the simplest case: two straight opposing lines through clear terrain. The second scenario shows the Red and Blue lines split by a mountain range. In addition, there is a weak point in the southern part of the Red line. When DBSCAN is run with liberal parameters (epsilon = 2, minStrength = 6), the southern part of the red line appears unbroken (Scenario 2A). When DBSCAN is run with more strict parameters (epsilon = 4, minStrength = 7), a gap in the line appears, and one of the cavalry units is shown to be isolated (Scenario 2B). Scenario 3A shows long lines through rough terrain. Scenario 3B shows the result of a Red parachute drop into the mountain pass that is part of the Blue line. The result is a break in the Blue line, and two isolated Blue units to the north of the pass.

One important thing to note is the dependence on the epsilon and minPoints / minStrength parameters. In the current version of DBSCAN these parameters are fixed. However, it may be desirable to automate the setting of these parameters based on game context (perhaps based on attack strength of opposing forces?).

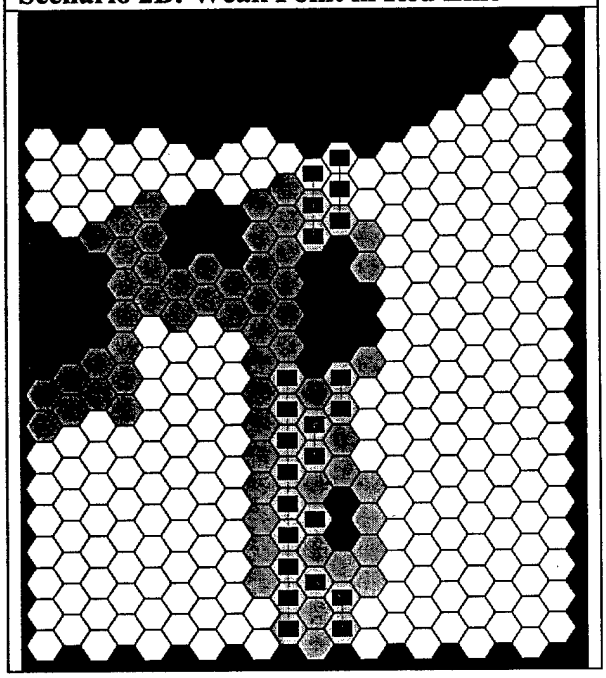
Scenario 1: Simple Lines with No Disruptions



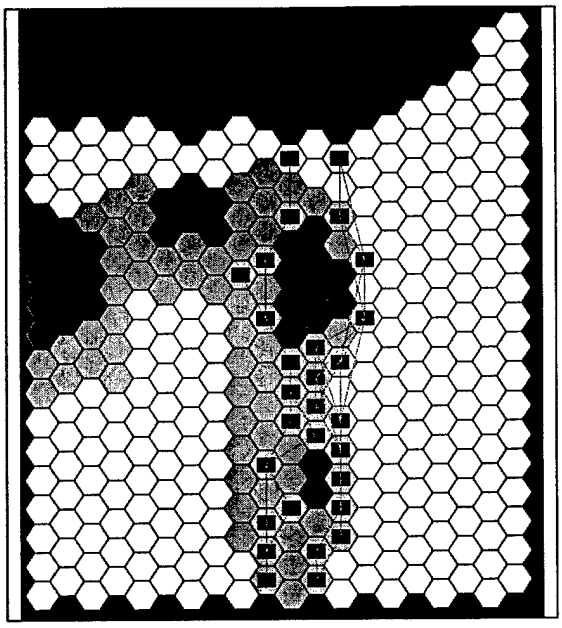
Scenario 2A: Lines Broken by Intervening Terrain / Distance



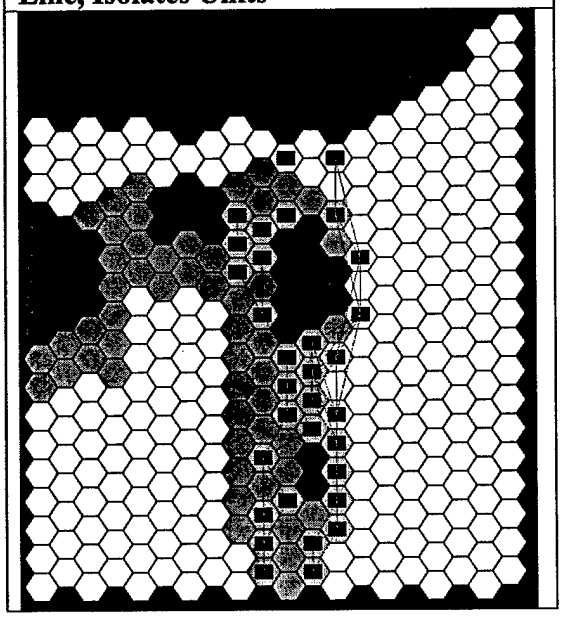
Scenario 2B: Weak Point in Red Line



Scenario 3A: Long Lines Through Rough Terrain



Scenario 3B: Parachute Drop Cuts Blue Line, Isolates Units



6.0 References

M. Ester, H-P. Kriegel, J. Sander, X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", Proc. 2nd International Conf. on Knowledge Discovery and Data Mining (KDD-96).

5.12 Graphical AI Technologies

Drew McDowell

Stottler-Henke noticed a need among game developers for a simple AI toolkit that would make AI design more approachable and efficient. SimBionic is the result of their development effort.

SimBionic provides a graphical user interface for designing game entity AI's, though it does not have to be restricted to game development. The AI designer visually draws out a representation of the AI in a dataflow manor using pre-coded conditions and behaviors. Once an entity's overall behavior has been described, the representation is 'compiled' down into a small format which the SimBionic library can read and interpret. The game developers then link their game code against the Simbionic library which then executes the compiled scripts in the context of the game.

The two key SimBionic advantages seem to be the graphical representation of decision flow, and the ease of re-combination of component parts. The graphical representation is quite nice and as the introduction points out, is perfect for representing your work to non-programmer collaborators. The component design approach allows content developers to generate customized AI behaviors in much the same way as a child builds a castle from differently shaped of wooden blocks.

To use SimBiotic, a programmer must develop a basic toolkit of simple actions built upon their provided scheduling, load balancing, and interprocess communication facilities. It may prove tricky to implement complex learning algorithms within this framework. However, once a general set of components are defined a non-technical content creator can combine them into new behavioral entities with relative ease. This requires a higher initial investment of effort, but allows for much more varied and interesting content to be generated for the game relatively quickly. Simbionic's visual debugger could also provide productivity gains because debugging work may be done at a higher level of abstraction.

The component organization approach that Symbionic uses appears to be similar to current 'operation oriented' data mining and expert system packages such as EvE, Eagle, Clementine, and Orange with a more highly developed user interface.

<Remaining text deleted because of proposal page limits. More information can be found in the final Phase I report.>

5.13 An Architecture for the Development of AI Avatars

John Tiller

1.0 Introduction

This report discusses an architectural approach to the development of AI "avatars" specifically designed for application with training games. The notion of an avatar extends the standard definition of artificial intelligence by adding specialized characteristics such as personality and the ability to interface with humans using speech, gestures, and facial expressions. The architectural approach taken here builds on the standard architecture of Evaluate-Plan-Execute for AI and provides a flexible approach for the development of AI avatars when permits the introduction of new technologies in a "plug-and-play" manner.

2.0 Outline

There are five major components to the architecture described here. These are shown in Figure 1 below. The components in this diagram are:

- **EPE** – The Evaluate-Plan-Execute AI engine.
- **Simulation** – The simulation engine.
- **CTM** – A Character Traits Module, described below.
- **HIM** – A Human Input Module, described below.
- **HOM** – A Human Output Module, described below.

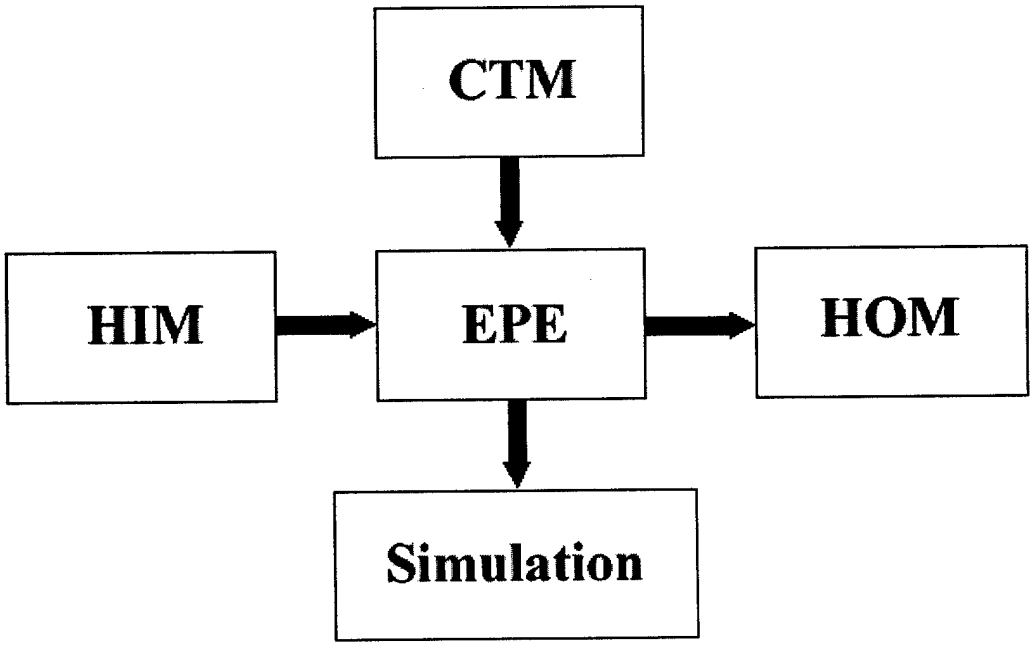


Figure 1. The overall avatar architecture.

In this approach, the various modules interact with each other through well-defined interfaces. This approach allows the specific implementation of any individual module to be variable as long as the interface remains the same. In this way, new technologies can be introduced into the architecture in a "plug-and-play" manner, thus greatly extending the useful lifetime on the approach.

3.0 Enhanced EPE Engine

The EPE engine associated with this approach would have to support not only the basic AI processing required to interface with the simulation, but also additional input and output to interface with the avatar components. These interfaces would be constructed along multiple channels of information as follows:

- **Channel 0** – Language channel.
- **Channel 1** – Expression channel.
- **Channel 2** – Emotion channel.
- **Channel 3** – Gesture channel.

That is, the basic communication channel that the EPE engine would have to support is that of language output and language recognition. Given the specialized nature of the environment, it would not be necessary to implement full language recognition, a challenging task, but rather only a limited extent of language processing based on the particular situation involved. For example, if the AI avatar was intended to implement a fighter pilot character, then the basic commands involving where to fly and what target to attack would be all that would be required. This may in fact be conveyed through the human selection of buttons and points on a map, or it could be more sophisticated through voice recognition of the commands.

The higher channels would then be used to modify that input to generate appropriate reactions by the AI processing. For example, when told to "Watch out!", a sense of urgency could be conveyed through the Expression Channel that would cause the AI to process this input with a higher priority than normal. If the human is also gesturing towards a particular location, then this could further influence the AI processing to look for information in that direction.

4.0 Character Traits Module

The purpose of the Character Traits Module is to influence the nominal execution of the EPE loops in the AI engine so that a type of personality to the AI processing results.

For example, by modulating the processing of the Execute phase of the AI engine, the resulting actions of the AI could vary from bold to cautious. Likewise, by modulating the Plan phase, the results could vary from Reckless to Pedantic. Finally, by influencing the Evaluate section, the AI could be seen to be foolhardy or tedious.

This approach allows a nominally constructed AI engine to be influenced in a way that varying types of behavior can result. By localizing this feature in an independent module and interfacing it to the EPE engine through an interface, each module can be developed independently and with maximum results.

The resulting CTM would have various levels of implementation based on the following:

- **National Characteristics** – Implements traits common to a group.
- **Personality Characteristics** – Implements traits specific to an individual.
- **Mood/State Characteristics** – Implements time specific traits.

That is, the design of the CTM module might proceed along these lines. First, it may be understood that the avatar is intended to be American. Second, the avatar could be designed to implement the personality of a fighter pilot. Finally, the time specific traits could be designed to reflect the varying feelings of that pilot in combat, such as anxiety or confidence. Putting these together then would result in the appropriate characteristics for an American fighter pilot in combat, suitable for implementation in the test-case described in the conclusion of this report.

5.0 Human IO Modules

There are two Human IO Modules in this architecture. One, the HIM input module, is responsible for taking human input and converting it into a form useful to the EPE engine. Likewise, the HOM output module, is responsible for taking output from the EPE engine and generating a representation of that output for a human player.

Each of these modules interfaces to the EPE engine through a well-defined interface. In this way, varying implementations of each module are possible. The basic approach to the HIOM interfaces is through "channels".

Each channel conveys a different type of communication. Any particular implementation of a Human IO Module may implement some or all of these channels. This allows implementations of varying complexity and sophistication to be design and built.

With respect to the Human Output Module, the following channels are envisioned:

- **HOM Channel 0** – Text and graphical output.
- **HOM Channel 1** – Voice generation.
- **HOM Channel 2** – Face animation.
- **HOM Channel 3** – Body animation.

Channel 0, text and graphical output, is the minimal level of support by any HOM. That is, the default HOM would cause text and graphical information to be generated from results conveyed to it by the EPE engine. At the next level of support, an HOM would be capable of generating spoken text from the same results. Moving up to higher levels of sophistication, an HOM would be capable of displaying a head, presumably human, and

animating that in appropriate ways based on the results from the EPE engine. Finally, at the highest level of sophistication, a complete body would be displayed and appropriate gestures and body language would be generated.

For example, in a given situation, the following results would be generated by the EPE engine.

- **HOM Channel 0** – "The hell with you!"
- **HOM Channel 1** – Shouting.
- **HOM Channel 2** – Angry Face.
- **HOM Channel 3** – Obscene gesture.

Depending on the level of sophistication of the HOM, the resulting output could either be simply the words conveyed on channel 0, or an appropriate level of shouting by voice generation technology, or combined with an animated face, or finally, with the full body language conveyed by channel 3.

With respect to the Human Input Module, the following channels are envisioned:

- **HIM Channel 0** – Text and mouse input.
- **HIM Channel 1** – Voice recognition.
- **HIM Channel 2** – Face recognition.
- **HIM Channel 3** – Body-language interpretation.

In the same way as output is handled, a particular implementation of an HIM would support one or more of these channels. For example, at the minimal level of support, an HIM would be able to convert text and mouse input from a human into appropriate input for the EPE engine. Likewise, at the next higher level of support, an HIM would be able to do voice recognition and convert that into an internal format. At the more sophisticated levels, the HIM would take picture or video input and be able to do facial recognition and body-language interpretation on that input.

6.0 Implementation

On possible development of such an AI Avatar architecture could proceed along two lines, one military oriented and one commercially oriented:

- F-16 pilot avatar – Based on the Microsoft Flight Simulator program.
- Risk board game avatar – Based on a computer version of the board game Risk.

In the first instance, the goal would be to implement a pilot avatar capable of expressing itself in standard military aviation terms. Likewise, it should be capable of accepting standard flight commands from a human through voice recognition.

In the second instance, a board game playing avatar would be developed that would be oriented towards more social interaction. This avatar would be capable of interfacing with a human player through more social interaction that would be common in a situation

involving a board game. Expressions of optimism, frustration, and other typical game playing interaction would be supported. It would also be possible to implement common facial expressions so that as the game progressed, the avatar would display the appropriate facial expressions and reactions for the current state of the game.

Finally, it would then be possible to integrate the F-16 pilot avatar implementation into a DARWARS compliant federation using the structure of HLA as described in the report "AI Agents within an HLA Federation", previously written by this author. With this, it would then be possible to invoke the pilot avatar as needed by the participants in the DARWARS federation depending on the need or desire for an avatar pilot to complete the training scenario.

5.14 Continuous Path Algorithms

John Rushing

1.0 Introduction

This report describes a series of experiments with continuous path optimization algorithms for planning flight paths. A flight path represents the path an aircraft would traverse when flying a mission. A flight path is assumed have a start point, target and termination point (which may be different than the start point or the same). The points are modeled using continuous x-y coordinates. The flight path is limited to a fixed number of segments. A set of hazards is defined in the same space. Each hazard has a location, a threat radius, and a threat rating. These hazards represent enemy air defenses such as surface to air missiles and AAA. The risk is assumed to be proportional to the time spent within the threat radius of the air defense system. Flying faster will reduce the threat but consume more fuel. An optimal flight path is one that minimizes the risk to the aircraft while satisfying fuel usage constraints. The objective of the continuous path algorithm is to find flight paths that minimize risk and satisfy fuel usage constraints.

2.0 Genetic Algorithm Formulations

Genetic Algorithms (GAs) are general-purpose search and optimization procedures. Genetic algorithms minimize or maximize an objective function with one or more parameters. The parameters must be coded in the form of a fixed length string of bits, which the genetic algorithm manipulates. The main component of the objective function for a flight path optimization is the sum of the threats posed by the enemy air defenses over the course of the flight path. A secondary factor is fuel usage, so a shorter path is preferred over a longer one with equal risk. A large penalty is applied for a path that consumes more than the available fuel. There are many possible ways for encoding a flight path as a bit string. In the experiments described below, three methods were explored:

- **Absolute Point Encoding:** The flight path is represented by a series of vertices. The vertices are specified by x-y coordinates on a grid. Each N bit section of the bit string specifies an x-y grid location. A larger N corresponds to a finer grid.
- **Relative Point Encoding:** The flight path is again represented by a series of vertices. The vertices are initially spaced equally along a straight path from source to target to exit. Each N bit section of the bit string specifies an x-y offset for a vertex from its initial position.
- **Relative Path Encoding:** The bit string represents a set of offsets or deltas. Each N bit section corresponds to a single x-y delta. The sum of the x deltas is normalized to the x distance required by the flight, and the sum of the y deltas is normalized to the y distance required by the flight.

3.0 Greedy Search Formulation

One issue with all of the GA formulations is that the distances, whether relative or absolute, must be encoded as fixed length bit strings. This limits the possible moves or positions, and is similar to using a placement grid, albeit of fine resolution. An alternative approach is to try moves of continuous size within some reasonable range. One way to do this is to use a greedy search procedure such as the following:

```
Do
  improving = false
  Pick a random distance D
  For each vertex in the path
      Try moving vertex N, S, E, W, NW, SW, NE, SE by D
units
      Evaluate cost of new path
      If (new path better than old path)
          Accept new path as best so far
          improving = true
      EndIf
  EndFor
While (improving)
```

4.0 Development

Three modules were developed to support the flight path experiments. The first component is a flight path evaluation library. The library is capable of reading and decoding bit string that represent flight paths, and then evaluating their cost given a set of hazards. Two programs were built using this library: an objective function program for use with a genetic algorithm, and a greedy path optimization program that implements the procedure described in section 3.

4.1 Software Structure

The software delivered with this report compiles into a library and two executables. The library contains a basic wargame model, and the executable contains the DBSCAN algorithm and a small command line driver for the algorithm. There is also a directory with a set of icons used by the program.

Module	Description	Classes / Functions
ContinuousPath	A library which contains methods for reading hazards and evaluating flight paths	Mission, Path, Threat, Point
PathCost	A program which takes bit string, decodes into a flight path, and returns evaluation	GA Driver Program
MinPath	A program which implements a greedy search to optimize flight paths	Greedy Optimization Driver Program

4.2 Building the Software

A Microsoft Visual C++ 6.0 workspace file (ContinuousPath.dsw) and three associated project files (one each for the library and executables) are located in the Source directory of the zip file. The libraries and executables can be built using these files. In addition, a CMakeList.txt file is provided so that with the use of the public domain CMake program, the source can be built on a wide variety of platforms.

4.3 Using the Executables

The continuous path programs are self-documenting. Running them with no arguments or the incorrect number of will prompt them to produce a usage message. All arguments are required for both programs. The syntax for the executables is explained below:

```
Usage: PathCost mission encoding bitstring
   mission:     name of the mission description file
   encoding:    encoding method for the bitstring
                  1: absolute encoding
                  2: relative encoding
                  3: relative path encoding
   bitstring:   bitstring for the flight path


Usage: MinPath mission
   mission:     name of the mission description file
```

Both programs read a mission file and write resulting flight paths as gnuplot command files. Gnuplot is a public-domain plotting package available at on the web from http://www.ucc.ie/gnuplot/. The mission description files have the following format:

```
Segments <#segments>
Source <x> <y>
Target <x> <y>
Exit  <x> <y>
```

```
Fuel Load <units>
Fuel Burn Rate <units>
Afterburner Rate <units>
Threat <x> <y> <radius> <factor> <factor in afterburner>
Threat <x> <y> <radius> <factor> <factor in afterburner>
...
Threat <x> <y> <radius> <factor> <factor in afterburner>
```

The numbers in the file are not tied to specific units, but it is assumed that appropriate combinations of units are used. For example, if distances are specified in nautical miles, and fuel load in gallons, then fuel burn rate should be in gallons per nautical mile. It is assumed by the model that the aircraft is either cruising (using fuel at the normal fuel burn rate) or using afterburners (using fuel at the afterburner rate) at any given time. The threats have different threat rates for the aircraft based on its speed.
While the MinPath program is meant to be run standalone, the PathCost program requires an external driver to generate the candidate bitstrings. In the experiments described below, a genetic algorithm was used as a driver. The GA is supplied on the zip archive in binary form: ITSC_GeneticAlgorithm.exe. It is a textbook GA, which uses steady state replacement without duplicates. The syntax for the genetic algorithm program is as follows:

```
Program: ITSC_GeneticAlgorithm

Options:
  -b <rate>             Bit mutation rate
  -e <evaluations>      Maximum number of unique obj fn
evaluations
  -g <generations>      Number of generations to run
  -h                    Print this message
  -n <num bits>         Number of bits in each chromosome
  -o <obj fn>           Name of the output image
  -p <size>             Number of population members
  -s <seed>             Random number seed

Description:
  ITSC_GeneticAlgorithm is a program that will run a
genetic
  algorithm on a specified objective function. The
objective
  function is implemented as a separate program
```

This can be used with the PathCost program as follows:

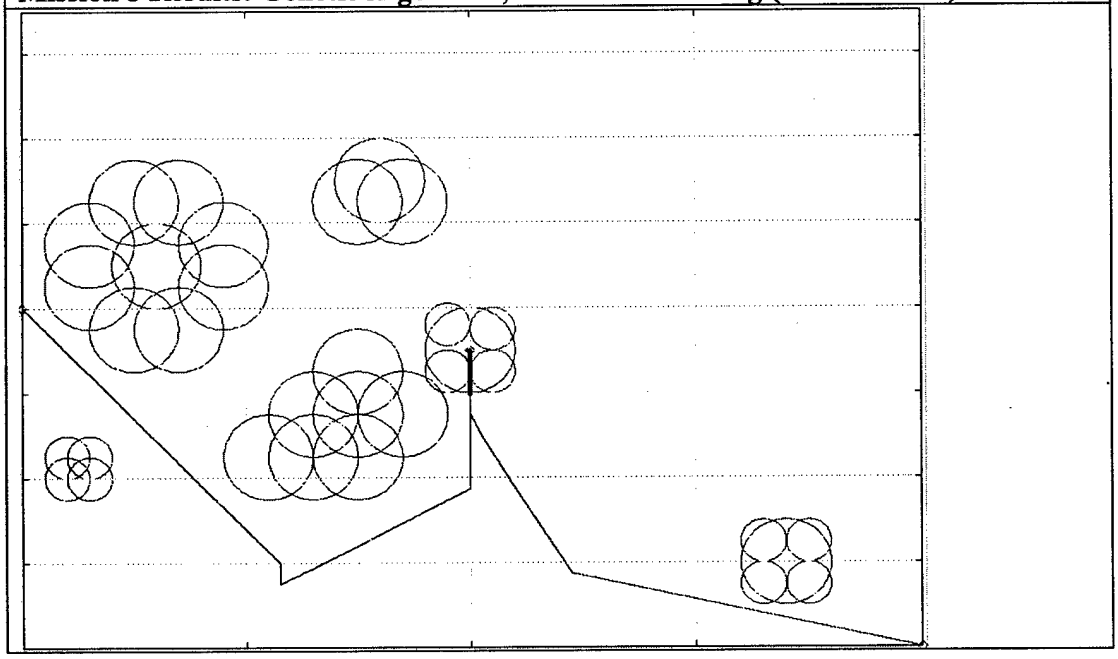
```
ITSC_GeneticAlgorithm -g 100 -e 10000 -n 240 -p 400 -b 0.05
-o "..\Source\Debug\PathCost Mission5.txt 3"
```

5.0 Sample Results

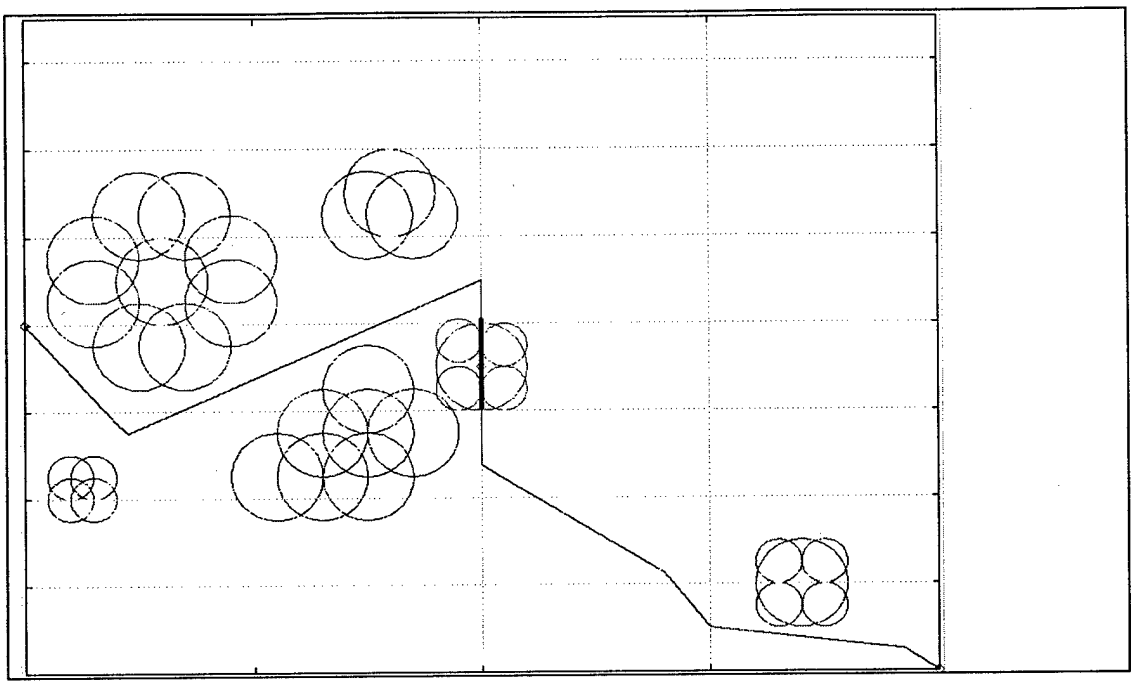
In the experiments described below, usage of afterburners was determined in a post-processing step. Flight paths were optimized using normal fuel usage rates, and afterburners applied later with any remaining fuel to reduce the effect of the threats. There are five sample mission files in the zip archive in the Mission* subdirectories. Each successive example is more complex than the preceding one. In the following figures, threats are indicated by red circles, the flight path by red line segments, and the source, target and exit by diamond symbols. The segments of the path where afterburners were used are indicated by bold symbolized line segments. All the figures were generated using Mission5 with different optimizations.

The genetic algorithm uses linear fitness normalization, and steady state replacement without duplicates. New population members were generated using crossover (75%) and bit mutation (25%). The results in the figures are the best result achieved using the particular encoding, considering GA population sizes of 200, 400, 800 and 1600 members and bit mutation rates of 1, 2.5 and 5 percent. The GA converged in all cases.

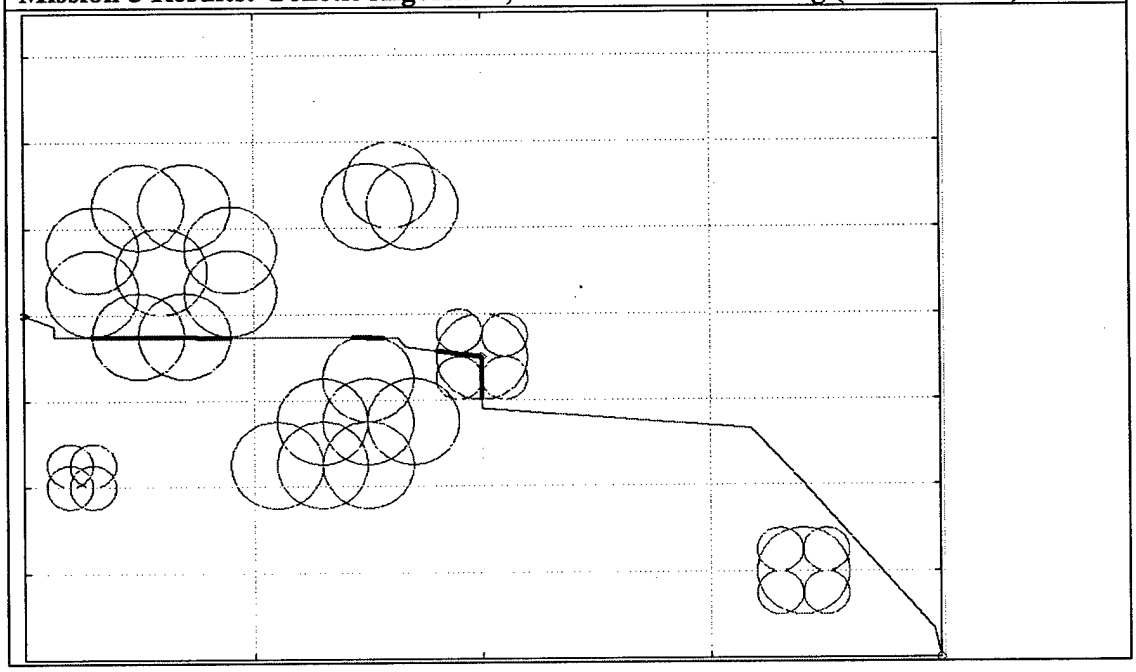
Mission 5 Results: Genetic Algorithm, Absolute Encoding (Cost = 12014)



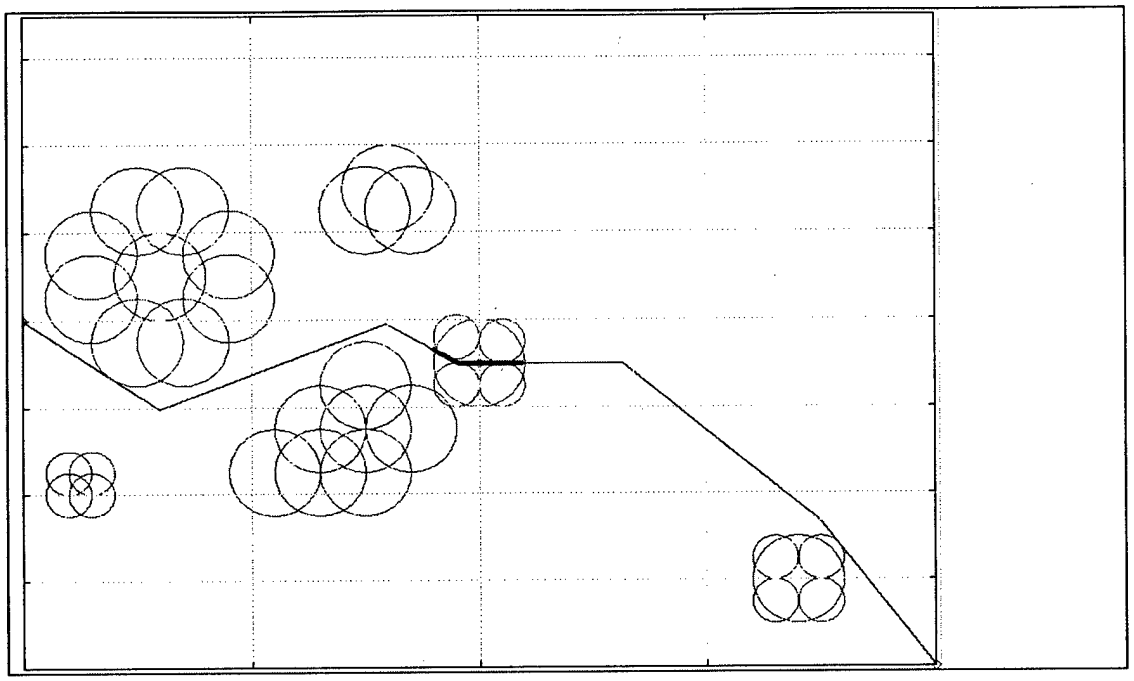
Mission 5 Results: Genetic Algorithm, Relative Encoding (Cost = 11868)



Mission 5 Results: Genetic Algorithm, Relative Path Encoding (Cost = 35135)



Mission 5 Results: Greedy Optimization (Cost = 12005)



As the figures above demonstrate, the genetic algorithm with absolute and relative encoding and the greedy algorithm all produce results of approximately the same quality. The genetic algorithm with relative path encoding fails to produce good results. This may be because the encoding does not easily allow good partial solutions to be combined to make a better solution. In relative path encoding, a change to one value changes the positions for the rest of the path because of the normalization. The genetic algorithms required relatively large population sizes (400-800 members) in order to avoid premature convergence to sub-optimal results.

5.15 Learning to Learn - Data Mining Applications for Wargames

John Rushing

Introduction

Wargames have a long history of use as learning tools for training of military personnel. One important aspect of this training is the analysis of player actions during the game. This analysis is used to reveal tendencies or potential weaknesses that the players themselves may not be aware of, so that they can improve their performance in future games and more importantly in real world decision making. Modern computer wargames are sophisticated software tools that are capable of providing detailed data about the actions taken by each player and the underlying situations within the game scenarios that lead up to these actions. There is a potential to use data mining tools to aid in the analysis of this data in order to improve the feedback provided to the player. The information derived by the data mining tools could also be used to provide information to the

wargame AI modules so that they can react in a way that will challenge the player in future scenarios.

Data Mining Queries

Data mining is a discipline that draws tools and techniques from many areas, including pattern recognition, statistics, database systems and others. The purpose of data mining is to aid in the analysis of very large data sets. In particular, data mining can be used to discover relationships between attributes in data sets, discover similar patterns, and perform classification and prediction. Information provided by data mining queries can be used for decision support. Data mining tools can be used to help characterize, model, and predict the behavior of human players. The information provided by these tools can then be analyzed for weaknesses or tendencies. This information can then be used to provide feedback to the player, and also to help devise or refine AI strategies to challenge the player in future games or scenarios. In general, data mining queries can be partitioned into two classes: directed and undirected. In the case of a directed query, the analyst has a particular goal in mind, or a particular question that they feel is relevant. An undirected query on the other hand may be used to identify relationships or patterns that may be of interest for further consideration. Undirected queries may reveal relationships that the analyst may not have considered.

Directed Queries – Modern Air Power

In order to make the preceding discussion more concrete, it is useful to consider the concepts of directed queries in the context of an actual wargame. Modern Air Power is a game which has potential for use in training for the USAF. In this game, players control aircraft, radars, SAM sites and other resources in an entire theatre of operations. Players must make decisions about when, where, and how to deploy the resources available to them. Some important aspects of the players' decision making processes could be captured by answering the questions such as these:

How does the player use SAM sites?
- Under what conditions are they active?
- Under what conditions are they dormant?

How does the player use tanker aircraft?
- How near do they go to enemy bases?
- Do they avoid enemy fighters?
- Do they fly in fixed patterns?
- Are they escorted, and if so how well?

When does the player willingly engage in dogfights?

How does the player select targets for ground attacks?
- Do they target enemy air bases first?
- Do they target enemy SAMs and AAA first?

How are multi-role fighters used?
- For ground attack missions?
- For air superiority missions?

Do they engage in attacks of opportunity? (strafing ground targets on return)
How aggressive are escort fighters?
Will they leave their charges to pursue enemy fighters?
Does the player counterattack when opportunities present themselves?

Undirected Queries – Modern Air Power

Another alternative is to frame the data mining queries in extremely general terms, with no specific targets in mind. This is the approach that is often used in association rule mining, where one is interested discovering relationships between attributes. In the case of wargames, the attributes may be user actions, events, game states, or other features computed from the game logs. The mining could be directed to discover patterns present at a particular time, or to discover temporal patterns. For example, association rule mining can discover rules of the form:

Whenever X1 and X2 and .... and Xn are true, Y1 and Y2 and ... and Ym are true also

Alternately, one might be interested in rules with time information, such as:

Whenever X1 and X2 happen followed by X3, player takes action Y

The temporal rules could involve simple sequences of events, or could also involve time windows.

Data Mining Techniques

This section provides a brief overview of three different classes of data mining techniques that may be useful in the context of wargames.

Clustering

Clustering involves grouping objects into classes so that similar objects are in the same class and dissimilar objects are in different classes. Clustering is also known as unsupervised pattern classification. Spatial clustering can be used to identify groups of supporting units, identify front lines, and identify gaps or weak points in lines. An example of using density based clustering for this purpose has been provided in a separate report (ref DBSCAN report). Clustering could potentially be used at a higher level to identify similar situations. In order to do this, a set of metrics describing the overall situation would be required. Such a set of metrics would include things like the relative strengths of the different sides, the locations of the objectives, the difficulty of the terrain, and possibly many other factors.

Supervised Classification and Prediction Algorithms

A supervised pattern classifier is trained based on a set of patterns with known class assignment. The classifier is later used to classify patterns where the class assignment is

unknown. Supervised pattern classifiers could be used for directed queries where a particular decision making process is being analyzed. In order to use supervised classifiers in this way two things are required: a set of features or attributes that contain information from which it is possible to predict the desired event, and training data containing instances of the event and near-miss non-instances. Feature selection techniques can be used in conjunction with classifiers to select attributes and improve classifier performance.

Association Rules

Association rules are capable of identifying relationships between attributes in large data sets. They can be used for decision support, classification and clustering. Association rule mining algorithms produce rules of the form:

$$(A1 \wedge A2 \wedge \ldots \wedge AN) \Rightarrow (B1 \wedge B2 \wedge \ldots \wedge BM)$$

The left side of the rule is known as the antecedent and the right side is the consequent. The rule simply states that whenever the antecedent is true, the consequent is often true as well. There are two parameters associated with the rule: support and confidence. The support indicates how often the rule applies (ie. the percentage of instances or transactions in the database where the antecedent and the consequent are true). The confidence is the probability that the consequent is true given that the antecedent is true.

Association rules could be useful in diagnosing tendencies of players. For example, the antecedents could be game conditions and the consequents could be actions taken by the player. If a particular rule is found that has high confidence, then that means that the player is consistently taking the same action when the same set of circumstances prevail. Using association rules will present challenges as it may not be simple to quantitatively characterize game situations and actions in the distilled form that association rule algorithms expect.

Some specialized association rule mining algorithms may be appropriate for the analysis of wargame data. Efficient methods for mining rules with item constraints make it possible to use association rule methods to derive a classifier (answer a directed query). In addition, temporal association rule mining methods may be used to discover temporal or causal patterns .

Derivation of Attributes for Mining

Data mining analysis typically involves discovery of relationships between attributes in large data sets. For many mining applications, particularly those related to scientific data or image analysis, it is necessary to extract high-level attribute information from the lower level raw data before mining. Often times determining the types of attributes required and methods for extracting them comprise a significant portion of the mining effort. Analysis of player behavior in a wargame will most likely require a significant attribute extraction process.

In some cases, multiple levels of analysis may be required in order to satisfy the data mining query. The results of a primitive or low-level query could be used as attributes or inputs for higher level queries. For example, if one is interested in analyzing when fighters are scrambled to intercept enemy aircraft, it may be necessary to first classify the flight profiles taken by the fighters. It is possible that the fighters might have been scrambled for some other purpose (such as fleeing). Another example is the use of multi-role aircraft. Multi-role aircraft can be used for different types of missions, and may even perform more than one mission simultaneously. Determining how aircraft are being used in a particular case is important for higher level queries that analyze the behavior of aircraft based on their roles.

Sampling and Completeness

For turn-based games, decisions are made during the player turn based on the situation at the start of the turn or phase of interest. For such games, temporal sampling of the data is unnecessary. For continuous time games such as Modern Air Power, it will be necessary to sample to derive patterns for the data mining process. Sampling may be uniform (collecting information at regular intervals) or may be based on events. Events could include both explicit player actions and things that might trigger a player response (such as enemy aircraft coming in range of friendly radar). The appropriate type and granularity of sampling will depend on the types of analysis being conducted.

One important factor in deriving information is the completeness of the input. It is important to make sure that there are many positive and negative instances of the phenomenon of interest. For example, if one is interested in predicting when a player will scramble fighters to intercept enemy aircraft, it is necessary to consider both the instances where the player did scramble fighters and those where the player did not. Note that it may not be possible to capture all of the relevant situations that factor into the player's decision-making process, since some such situations may not have arisen during the scenarios in which the player has participated. This means that the rules derived may be incomplete or overly simplistic and not reflect the full complexity underlying the player's decisions.

Example Query

In order to make the ideas discussed above a bit more concrete, an example query related to the Modern Air Power game will be considered. We will attempt to formulate a data mining query to answer the following question:

> When will the player scramble fighters from a base to engage the enemy?

Attributes

The attributes available for predicting the player behavior depend on the level of information captured in the game replay logs, and the amount of information available to

the player. Depending on how fog of war settings are chosen and implemented, the player may not have exact knowledge of the situation. For example, consider the following set of attributes that could be used:

Number of enemy air superiority fighters within 50 miles
Number of enemy air superiority fighters within 100 miles
Number of enemy ground attack aircraft within 50 miles
Number of enemy ground attack aircraft within 100 miles
Number of enemy support aircraft (tanker, AWACS, etc.) within 50 miles
Number of enemy support aircraft (tanker, AWACS, etc.) within 100 miles
Number of friendly air superiority fighters available for intercept missions
Number of friendly air superiority fighters airborne within 50 miles
Number of friendly air superiority fighters airborne within 100 miles

Some of these attributes might have to be modified or combined. For example, the player might know that there are incoming aircraft at a given range, but might not be able to distinguish ground attack aircraft (like an A-10) from air superiority fighters (like an F-15) without visual identification. The division into 50 mile increments is arbitrary, and it may be better to use finer increments, more increments, larger range etc.

There are of course many other factors that might be considered, such as:

Quality of the aircraft
Quality of the pilots
Value of targets in the area (which may be difficult to quantify)
Course, altitude and speed of enemy aircraft

The player's decisions may be at least partially governed by factors that are not captured by the attributes. Furthermore, the decisions may not be entirely consistent, as the player may make different decisions in similar situations (perhaps consciously trying to take the enemy by surprise).

Analysis

The goal is to predict from the derived attributes when the player will scramble fighters to engage the enemy. This is a two-class classification problem, and therefore any classifier may be used to solve the problem, including ensemble classifiers. Using a white box classifier such as a decision tree may be most appropriate since the models learned by such a classifier can be directly understood and analyzed. If a black box classifier such as a neural network is used, then feature selection methods may be useful for reducing the number of inputs required, improving classifier accuracy, and providing information about which attributes are most relevant to predicting the behavior of the player.

Sampling

In order to train the classifier, it is necessary to provide both positive and negative instances of the phenomenon of interest. In this case, it is necessary to provide instances

where fighters were scrambled, and instances where they were not. Uniform time sampling could be used to provide the initial data. If the number of points produced by the uniform sample is too great, then event triggers based on enemy aircraft coming within some distance thresholds could be used instead.

5.16 The Use of AI Technologies in Wargame Development

Drew McDowell, John Rushing and Steve Tanner

Artificial Intelligence Techniques for Wargaming

Over the past several decades, a great deal of research and development has gone into creating new ways to simulate intelligence in computer systems. Early on, it was assumed that it was only a matter of time before we could create machines that either think, or could simulate thought closely enough to fool any human. This goal has clearly not been met. However, several of the techniques being pursued have yielded tangible and productive results, if applied properly. But how does one know how and when to apply these techniques properly? This paper addresses this for the domain of wargaming. Section 2 provides a general overview of a wide range of what are considered to be AI technologies. Then, Section 3 describes how, when and why these should be applied to wargames and training systems.

The basic reason for looking to AI technologies in war games is fairly straightforward: They can be used to improve game efficacy and can also be used to improve the general training experience. For example, several modeling techniques can be used to better simulate different opponent behavior, mimicking the actions of known enemies. It can also improve the skills and abilities of opponents, such as more efficient use of limited resources (e.g. continuous path planning for planes). Some techniques, such as data mining, can be used in post processing of game results to find both strengths and weaknesses of individuals and teams, thus providing a feedback mechanism that will help improve training effectiveness.

Technologies

Agents

An intelligent agent can be thought of as an independent unit of software that views its environment through a series of sensors and acts upon it through actuators. In other words, the agent is aware of its environment, and reacts within that environment. The internal decision making process can range from complex to simple and they may interact with their environment and other agents in any number of ways. This approach can give rise to complex emergent behaviors that mimic what occurs in the field. Simple rules at the agent level can yield complex emergent behavior for a larger group. This makes the problems of group dynamics easier to model, but harder to prove. Agents can also act independently, and so can be used to help modularize the code, allowing a more plug-and-play system. Web-services are an example where different organizations provide different services as agents, without knowing too much about one another.

Although agents are really a design paradigm rather than an AI 'technology', they are a very powerful approach to some problems. An agent based approach allows for a more compact, organic design, that more closely imitates the target being simulated. An agent based system is able to model unit behavior, which is important for some modeling situations, especially where non-determinism is required. For example, modeling unit behavior on a battlefield may more realistically simulate real-world command structures and communications mechanisms, especially in chaotic and rapidly dynamic situations where troops must act in a more autonomous manner.

Blackboard Systems

Like agents, blackboard systems can be thought of as more a design paradigm rather than a new technology. Specifically it is a way for different processes to communicate with one another, and exchange information. The basic idea is that all processes are able to view the same "blackboard", an area in which they can both post and read information. If a process needs information, it can post a request to the blackboard. If another process can provide this information, or even a portion of the information, it responds by posting it. In this manner, multiple processes may be involved in generating all of the information originally requested. It is the idea that a given process will provide only what it knows, that this information can be combined into a larger whole, and the processes need not know about or be integrated with one another, that provide the blackboard approach with its power.

Classification and Prediction

Classification and Prediction techniques can be used to codify how a system should behave during known (and sometimes unexpected) events. In general Classification is the process of describing patterns, taxonomies or attributes associated with data or actions. For example, a Fault Tree that is used to describe the possible faults that a given system can exhibit is a form of classification. A given classification system can then be used to help with prediction and planning, for example through the use of induction and deduction based upon a given Fault Tree.

Decision Tree Induction

Decision Tree Induction algorithms create a flow chart of decision nodes that partition data samples into classes. The most well known of which is the ID3 family of algorithms that greedily partition sample data into classes based on the information gain of splitting the unclassified samples on untested attributes. Decision Trees work best when a domain expert is available to provide information for the trees and to verify the resultant data structure. Once such a structure is created, the inductive traversal of the tree is very fast and efficient, making them ideal techniques to use in a well defined domain when speed an important factor. However, they tend to be brittle, meaning that they tend to break down when faced with information outside of known paths. They do not handle unknown situations very well. One example of their use is a Failure Modes and Effects Analysis (FMEA) tree. Such a tree can be used to either deduce the cause of a problem, or to induce the likely effects of a fault.

Bayesian Classifiers

A Bayesian classifier is a statistical classifier that calculates the probability of a sample's membership in a certain class. It does this via a straightforward application of Bayes's Theorem. This approach compares well with other classification methods and can, in theory, have the minimum error rate among any classifier. As an example of their use, these types of classifiers have recently become popular as junk mail filters.

Neural Networks

Modeled after biological neurons, neural networks are a collection of interconnected independent units that classify samples by passing their attributes through a series of weighted input and output layers. A trained network can be computationally very efficient, but it is difficult to inspect the network state and gain any information about what knowledge it contains. Neural Networks are appropriate for situations where outcomes are known but difficult to model mathematically, such as non-linear systems.

Genetic Algorithms

The use of Genetic Algorithms is an approach to problem solving that is modeled after natural evolution. A solution to a problem is encoded as a single organism which may undergo reproduction (through cloning or crossover) and mutation. Groups of such solutions are put into populations where they compete with one another based upon a given fitness function. The most-fit members of the population are allowed to reproduce and a new generation is created. Genetic Algorithms are computationally intense and it is sometimes difficult to define an objective fitness function. However, they can be very useful when there are no known heuristic approaches for solving the given problem, and often develop unexpected and clever solutions.

Association Rules

Association Rule mining is a statistical process of discovering relationships between events in a given set. This is not the same as discovering causality. It is only the discovery that the two events are statistically linked in some manner. The rules are generally of the form **A -> B** where the consequent is said to occur with the antecedent and the whole rule has some known support and confidence level. In general this technique offers more mathematical rigor than some of the other classification approaches, since it relies so heavily on straightforward statistical analysis.

Clustering

Clustering techniques are often used to find patterns in data. For example, such techniques could be used to identify patterns associated with frontlines and their weak or strong points, thus making them useful in tactical planning.

Hierarchical Clustering

Hierarchical clustering works by grouping data points into a tree of clusters. The tree can be constructed from the bottom up or top down, but the central theme is to group 'nearby' data points into sub-groups (internal tree nodes) and iteratively build a tree like structure until the desired number of partitions is achieved.

Density Based Clustering

Density based clustering algorithms generally grow regions of the data space into clusters, if they meet some sample density requirements. These algorithms are able to discover arbitrarily shaped clusters by joining regions of dense samples in the data space, and forming boundaries at low sample density regions.

Case Based Reasoning

Case Based Reasoning systems store knowledge of prototypical scenarios in an abstract symbolic description. When a case based reasoner is used to recognize a scenario, it first checks its stored knowledge base to determine if it has seen a case just like this before. If such a case is found, the reasoner returns the known solution. If an identical case is not found, the reasoner will search for cases with close similarities and propose a solution that is a combination of the nearest known solutions. Such a system therefore tries to fit the facts to its known set of cases. For example, such a system could model known real-world scenarios as a set of cases. Then, when confronted with a new situation, it will react based upon the closest match to these known scenarios.

Constraint Satisfaction

The basic idea for Constraint Satisfaction is to search through a solution domain for a set of values-bindings that will satisfy a series of constraints. A constraint is a simple logical relation among several attributes of the search space that represents some partial information that is known about the given problem. Typically, the constraints are listed declaratively and the CSP system will bound the search space and attempt to find bindings for the variables presented. CSP systems are an appropriate technique to use when trying to satisfy multiple conflicting or orthogonal constraints.

Emergent Behavior

Behavior that results from a set of very simple rules when applied to a complex environment is said to be Emergent. All AI approaches could be said to result in emergent behavior, but this term is usually reserved for behavior that results from actors that explicitly do not utilize models of their environment.

Automata

In computer science, Automata refers to a very simple computational unit that senses some input and changes to a new internal state based on simple internal rules and a limited history. It may be helpful to visualize them as a flow chart with a finite number of nodes. At each node, or state, the automata senses its input and moves its internal state to the next node. It is important to note that the automata is restricted to local information such as which state the automata is in and the states of its neighbors. It is also important to note that groups of such simple machines are capable of highly complex behaviors, as detailed in Stephen Wolfram's *'A New Kind of Science'*.

Swarm Theory

Swarm theorists assert that social interaction optimizes cognition. The swarm approach is very similar to that of genetic algorithms, but comparison and imitation are the driving mechanisms rather than competition and reproduction. Individuals within the swarm compare themselves with members of their neighborhood and alter their behavior to be

more like their 'best' neighbor. The resulting self-organizing behaviors are an emergent property that can be applied to a wide variety of search and optimization problems.

Pathing Algorithms
Pathing Algorithms can be thought of as a type of informed search. For example, what is the shortest (least expensive) from one node on a network to another. Such algorithms are a critical component of many route planning systems.

Djikstra's Algorithm
Djikstra's algorithm for shortest path graph traversal is an optimal approach for discovering the shortest path between two points on a connected graph. The algorithm builds a shortest-path tree from the source node to every other node in the graph until the shortest path to the destination node is found. A side effect to this algorithm is that the shortest paths from the source node to every other node in the graph are discovered along the way.

A*
The A* (pronounced A-Star) algorithm is one of the classic informed search algorithms. The algorithm progresses through a tree-like search space by tentatively moving ahead one node and calculating the estimated distance from the new cell to the goal. This tentative step is then put into a priority queue that is sorted by the shortest cost of the path to the point + the estimated path cost to the goal. This is repeated until the goal node is reached.

Planning

Markov Decisions Processes
In Markov Decision Processes sequential decisions are decided upon in terms of projected risk vs. reward. Projected reward is based on a series of conditional probabilities that represent what information is available to the model. A decision is then made by applying a general policy of action that can be optimized as the model moves through each state and gains experience.

Computing all of the possible states and rewards can be computationally intense and thus these decision processes are often modeled as a series of Bayesian Networks called a Dynamic Decision Network. This approach is robust enough to deal with partially observable environments, plan revision, and unexpected observations.

Expert Systems
Expert systems are codified decision processes created by an expert who carefully studies possible scenarios of actions and consequences and details how the system should react. This approach yields optimal responses tailored to the information available and the preferences of the creator. The quality of the decisions is highly dependent upon the information contained in its knowledge base and thus can break down if presented with unexpected situations.

AI In Wargames

Wargames can be extremely complex systems, with several, possibly conflicting goals. The game must hold the interest of the player, meaning that the game must seem new or different each time it is played. The game must be able to deal with multiple skill levels of players, and should help a player improve over time. Furthermore, in the context of training, the game should supply post-play analysis on how a player or team of players performed. Toward these ends, AI technologies can play a vital role in the next generation of war games. They can be used to create a more realistic gaming experience, reducing or eliminating the "canned" feel of many gaming scenarios. They can adapt to a user, creating a new feel each time the game is played. They can also monitor a player's progress and provide insight and suggestions on a player's strengths and weaknesses.

In general, AI techniques can help Wargames in 3 primary areas: Added realism, Improved performance, Post-game analysis. Note that Improved performance is not constrained to simple speed improvements, but also includes more intelligent planning and execution.

Software Agents

Agent technology would be useful for modeling the behavior of independent actors in wargames. Since an agent is only privy to local information, they can be used to model the behavior of individual war fighters or groups, allowing for a more realistic simulation. The control logic within an agent can also be customized so that each agent could behave differently allowing for many combinations of reactive and scripted actors in a scenario. This would mean that the game would play differently each time, and the outcomes would be much more non-deterministic in nature. This approach was used to great success by MASSIVE Limited (http://www.massivesoftware.com/) when it was called upon to simulate the large scale battle scenes in the popular 'Lord of the Rings' movies.

Hierarchical Agents

A series of agents could be used to model the hierarchy of a military command. In such an agent-based chain of command, each level would be privy to more information and command over more units as it moves up the chain. These commanding agents need not be present as units on the field, but could operate behind the scenes. For example, such agents would be able to view the state of a sub-section of the game board and be able to communicate via blackboards with units directly above and below them in the chain of command.

These agents would be tailored to the specific jobs they perform. Agents in charge of a few squads could focus more on tactical decisions, whereas battalion and theater commanding agents would be focused more on strategic planning.

Sample Strategic Agent: The Strategic agent could synthesize any information that has been passed up from field commanders into a probability map (e.g. a Belief network) of the state of its arena of control. Once these probabilities are assessed, the strategic agent could form goals and optimize force deployments for offense, defense, and reserves.

From this knowledgebase, plans can be built for each of the Tactical agents and passed down the chain of command, delegating the tactical decisions to these lower level agents.

Sample Tactical Agent: A tactical agent could look at the subset of a battlefield (say a 16x16 area) that it has influence over, and attempt case-based reasoning by classifying this view as most similar to one of a few dozen prototypical cases with known response strategies. These may have been designed by an expert to mimic the combat styles of a specific opponent, or could be more general in nature. The agent would then carry out the maneuvers associated with this prototype. These response maneuvers could be any one of a set of maneuvers which are chosen based upon its standing orders handed down from its superior agent in the chain of command.

It is also feasible that other types of agents would be developed for mid-level command chains that facilitate communication between the strategic command agents and the field leaders. These agents could perform other duties from logistic support, to coordination with air/sea forces, to sub-goal planning.

It is important to remember that any agent along the chain of command could be fully autonomous or fully scripted to add realistic, interesting, and unpredictable effects to the overall behavior of the fighting force.

Emergent Behavior

Emergent Behavior models such as Swarming and Automata can be applied to wargames as a means to simulate the behavior of a relatively large number of entities, such as individual troops. Each entity would be represented by a separate, simple process. Each of these processes would have a very limited set of rules to follow. The result would appear to be complex emergent behavior, similar to how a real set of inidivuals would behave. For example, the cellular automata approach to a combat force could be modeled by giving each unit a simple rule system to react to various stimuli from the surrounding environment such as:

If there is a higher ranked friendly unit in view, move nearer to it and accept orders.

If there are no enemy forces visible, move towards the nearest goal.

If there are less enemy forces than friendly forces , engage the enemies.

If there are a few more enemy forces than friendly forces, dig in and take cover.

If there are many more enemy forces than friendly forces, retreat.

Simple rules such as these can yield surprisingly complex behavior (XXX I need a good Wolfram quote). By incorporating social interaction and weighing it against individual goals, one can model a very robust and natural feeling behavior: In this case, squads would begin to form around leaders, weak defenses may be exploited, and front lines may become entrenched.

Intelligent Prediction and the Use of Classifiers

If a process within a game has access to all knowledge (e.g. its fog-of-war is off) and has a robust enough model, then it can perform substantial and accurate predictions of an

opponent's behavior. Such access to knowledge is unrealistic in the real-world, but can be used in a gaming environment to enrich the playing experience and for the training potential it offers. There are a number of AI techniques that can be brought to bear on such predictions.

Decision trees and Bayesian classifiers can be used in a wargame environment to quickly decide the general state of a game and the next appropriate action. Such methods are a good choice to implement canned and known scenarios. They offer a way to codify strategeies and tactics, they are fast, and they are logical. These are the methods that several chess programs use to decide next moves during the opening and end game portions of play.

These methods do, however, generally produce consistant results. This means that a human opponent can, over time, deduce these codified strategies and tactics and figure out solutions to thwart them. In some cases this is exactly what should happen, and is part of the learning experience desired. However in most cases, the game should be less predictable. One approach to deal with this is to use multiple decision trees or classifiers and randomly switch between them. This offers the advantages of these approaches while working around some of their shortcomings.

Another benefit of such classifiers is that they can be updated and pretictions of outcomes can be recomputed each turn. Therefore, it is possible for the system to be incrementally improved over time. It can also persist between game sessions so that the most effective approach for dealing with specific opponents can be used. In other words, such a system can tailor itself to for each opponent it faces.

In addition, data mining techniques could extract profiles of the probability-of-action for human player based on previous games. This information would be a significant advantage in predicting possible game states during the prediction and policy building stages. For example, this information could be used to bound the search space, quickly eliminating unproductive avenues of reasoning.

Evolutionary approach

Control sequences for units can be developed and improved through the use of evolutionary approaches. Each type of unit could be evolved through simulated games of the computer playing experts or other computer opponents. These games would serve as the competition portion of the evolutionary process, continually improving the average performance through elimination of the poor performers, rewarding the good performers, and innovation through crossover and mutation. Multiple evaluation approaches could be proposed: Evaluation by accumulating victory points, evaluation by human experts, evaluation by comparison to actual data, etc…

Opponent Modeling

An expert player is observed. His game history is used to train a learning system (decision tree, neural net, etc…) that will behave similar to this expert in subsequent games. Note that the game history may not be from actual game play, but from historical

studies off actual engagements. If sufficient information were present, it would be feasible to simulate the strategies of known personas in a much less brittle form than the static scripting methods used presently.

Player evaluation
Many of the classification algorithms may be used to mine post-game data to determine strengths and weaknesses of a student's performance. This data can be stored and used in future sessions to exploit a student's tactical and strategic weaknesses or predict his plans, allowing the AI to better fulfill its role as opponent and teacher.

5.17 Bibliography

The Art of Prolog, Sterling and Shapiro, MIT, 1986.

Bayesian Networks and Decision Graphs, Jensen, Springer, 2001.

Data Mining, Han and Kamber, Morgan Kaufman, 2001.

Data Mining Solutions, Westphal and Blaxton, Wiley, 1998.

Genetic Algorithms, Goldberg, Addison-Wesley, 1989.

Introduction to Game Theory, Morris, Springer, 1994.

Numerical Recipes in C, Press, et al, Cambridge University, 1988.

Neural Network Architectures, Dayhoff, Van Nostrand Reinhold, 1990.

Practical Optimization, Gill, Murray, and Wright, Academic Press, 1981.

Smalltalk-80, Goldberg and Robson, Addison-Wesley, 1989.