12-1993

Developing Realistic Behaviors in Adversarial Agents for Air Combat Simulation

George S. Hluck

AFIT/GCE/ENG/93D-06

Developing Realistic Behaviors in Adversarial Agents for Air Combat Simulation

THESIS
George S. Hluck
Captain, United States Army

AFIT/GCE/ENG/93D-06

Approved for public release; distribution unlimited

AFIT/GCE/ENG/93D-06

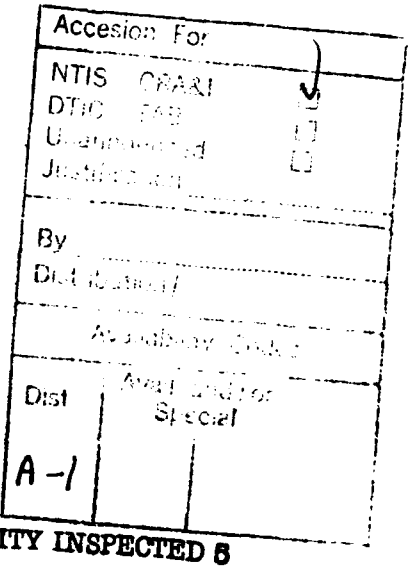
Developing Realistic Behaviors in Adversarial Agents for Air Combat Simulation

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

George S. Hluck, B.S.
Captain, United States Army

December, 1993



Preface

I would like to thank my thesis advisor, Major Gunsch for his guidance and mentorship. His high standards and expectations for course and research work were greatly appreciated. I would also like to thank my committee members, Dr. Hartrum and Dr. Santos, for their time and effort.

George S. Hluck

Table of Contents

List of Figures

List of Tables

Abstract

This thesis describes an initial effort into creating a rule-based, reactive system for air combat simulation. This program uses the object-oriented extension of the expert system tool known as the C Language Integrated Production System (CLIPS). This effort rose out of the need for creating and integrating semi-autonomous forces for the Distributed Interactive System (DIS).

This thesis describes the basic maneuvers a pilot uses in present air-to-air combat. The methodology includes the design decisions, knowledge-base development, phase architecture, and maneuver architecture development. The actual implementation of the selected architecture is described. This thesis also discusses the results of experimental runs with two agents maneuvering against one another.

Developing Realistic Behaviors in Adversarial Agents for Air Combat Simulation

I. Introduction

1.1 Background

The Air Force is constantly searching for economic and efficient ways to train pilots. One manner in which this may be achieved is through the use of computer simulations to test and train pilots. The difficulty lies in implementing an effective training system which can link together many forces from various units into one virtual, combined-arms battlefield. Not only does the system have to incorporate a mixture of actual units, but also create additional semi-autonomous and autonomous (virtual) forces. The advantages of this system would enable a few pilots to train with parent and combined-arms units without having to involve the presence of additional personnel.

The Advanced Research Projects Agency (ARPA) is currently promoting and funding research for a Distributed Interactive Simulation (DIS). DIS would enable a multitude of units to train together on one virtual battlefield without having everyone physically at one location. For example, on this battlefield a tank commander at Fort Knox, Kentucky could look out of his hatch and see an F-16 piloted by someone at George AFB.

There is great need for an expansion of computer-generated forces in the military. Organizers of the 3rd Computer Generated Forces Conference briefed the attendees on present shortcomings (14). There presently is no balanced representation of joint operations in military computer simulations. One major shortcoming is the minimal Air Force participation in this field.

Some of the artificial intelligence (AI) personnel at the Air Force Institute of Technology are investigating ways to implement semi-autonomous computer generated forces. We are presently attempting to expand on a former class project which utilizes adversarial agents in an air combat simulation called MAXIM, created using the Common Lisp Object System (CLOS) (8). The simulation pits two agents against each other in simple air-to-air combat. The simulation is primitive. The plane agents follow simple flight paths and use simple strategies. The challenge exists to expand MAXIM to a simulation which can interface with other simulations and realistically perform.

1.2 Problem

MAXIM, in its present form, is a simple reactive system. It avoids planning altogether, and does not anticipate and select an entire action sequence before acting. Reactive systems are normally associated with robotic systems, since robots normally have to deal with dynamic situations. An intelligent system that operates in a moderately complex or unpredictable environment must be reactive. In being reactive the intelligent system must decide when to start thinking, when to stop thinking, and when to act.

Captain Dean Hipwell and I started working together on a similar idea in a class on knowledge-based systems. Our thesis efforts began as a class project, and have greatly expanded since its conception. We worked together on the design of the system and concentrated on different areas for implementation. I essentially worked on the maneuvering, one-on-one aspect of the problem while Captain Hipwell concentrated on the cooperative, two-on-two aspect of the problem. Captain Hipwell was also the primary author of the flight model (13).

We believe it is feasible to create a rule-based, reactive system where agents participate in air-to-air combat. We used an expert system tool known as the C Language Integrated Production System (CLIPS), and eventually progressed to using an

object-oriented extension known as the CLIPS Object Oriented Language (COOL). We call our program Pilot Decision Phases in CLIPS (PDPC). PDPC is a rule-based reactive system where objects act as autonomous agents who can maneuver in an air combat situation.

1.3 Research Objectives

1. Define and create a phase architecture.
   - Identify the phase transitions of a pilot while executing a mission.
   - Decide how an agent can transition between phases.
   - Decide how to correctly implement such a structure in CLIPS.
2. Define and create a maneuver architecture.
   - Examine in detail air combat maneuvers and the tactics involved in their execution.
   - Decide how to decompose a maneuver into parameters so that a flight model can use the parameters to fly an agent.
   - Decide how to correctly implement the architecture in CLIPS.
3. Use knowledge acquisition techniques to encode a set of maneuvers.
   - Find appropriate knowledge sources for use as domain experts.
   - Transfer the information by encoding the maneuvers into a rule-base.
   - Verify and validate the knowledge base.
4. Enable an agent to use an opponent's history to predict where he will fly.
5. Enable an agent to use AI techniques to plan while it has time.

1.4 Summary of Approach

1.4.1 Phase Architecture. A successful air-combat mission normally consists of a sequence of phases. During each phase the pilot needs to accomplish certain

tasks before moving on to the next phase. The phase transitions are not always sequential; sudden environmental changes can prompt the pilot to switch to another phase to execute an appropriate maneuver. The same approach is used in PDPC. in PDPC. A complete sequence of phases were created where an agent's behavior in this system is controlled by this phase architecture.

1.4.2 Maneuver Architecture. Traditionally, the information in a reactive system flows from the environment to the object (in this case the agent) via sensing. The information then flows back into the environment via action done by the agent. As more complex behaviors are introduced into the system, this loop can take too much time. I attempt to find a balance; some way to layer the appropriate behaviors so the system can essentially react as quickly as possible. A simple architecture would consist of two vertical layers: perception and action. The agent would receive some input and then react to that input, if necessary.

My approach uses an architecture which consists of 3 vertical layers: perception, reactive plan selection, and action. The perception layer receives input and feedback from the environment. It outputs a selection to the reactive plan selection layer. This layer then outputs a sub-goal for execution to the action layer. The action layer executes the action, and outputs feedback to the perception layer. Since it is desirable for the architecture to approach real-time performance, anything more complex can become computationally inhibitive.

1.4.2.1 Perception. The perception layer receives inputs from several sources: feedback from its action, actions from opponents, and actions from cooperative agents. Although I am not concentrating on the cooperative aspect, it is still important to incorporate this input.

1.4.2.2 Reactive Plan Selection. This layer contains the maneuver plans. It will select an appropriate maneuver for execution. This plan is the input

for the action layer. This is also the layer where the agent, in the future, will be able to 'plan while it can' by activating something similar to an anytime algorithm.

1.4.2.3 Action. This is where the goal is given to a flight model and a new location is calculated for the agent. The state of the simulation is also updated at this time.

1.4.3 Maneuver Implementation. The bottom line in airplane maneuvers is maintaining energy, both potential and kinetic (31). An agent must maneuver to gain an advantageous position while still maintaining energy.

My goal was to implement the following set of complex fighter maneuvers:

- Pursuit curves
  - lead
  - lag
  - pure
- Lag displacement roll
- High yo-yo
- Low yo-yo
- Lead turn
- Nose-to-nose turns

The presence of cooperative agents compounds the difficulty of this implementation. A leader, when maneuvering, needs to make sure he and his wingman do not collide. The problem multiplies while flying in a large formation. An agent cannot blindly execute a sequence of maneuvers without checking for other cooperative agents in the area. PDPC attempts to avoid collisions with the appropriate rules.

1.4.4 Algorithmic Planning. Although not implemented, we realized the need for the addition of some sort of embedded planner. While not in a confrontation an agent should be able to do minimal planning on its own. I think it is possible to implement an appropriate algorithm to enable an agent to accomplish this. Anytime algorithms can be implemented by interleaving computation and action. This concept neatly fits into a reactive system for air-combat simulation since there are lulls in engagements in which an agent can adjust its plan or plan for a future situation.

1.5 Assumptions

I made two major assumptions at the beginning of these research effort:

1. The developed flight model is sufficiently realistic for research use.
2. For every possible situation in air-to-air combat a maneuver exists, which, if properly executed, results in appropriate behavior.

1.6 Scope

This research effort concentrated on the development of a phase and maneuver architecture for a rule-based reactive system. To develop a complete system, where the behavior of these agents is indistinguishable from those of actual pilots, is beyond the scope of this research effort. PDPC can presently run rudimentary simulations. The rule-base contains an appropriate amount of knowledge which enables the agents in the system to maneuver as virtual pilots and show that potential exists for this approach.

1.7 Summary

The completion of this thesis effort has enabled me to create a rule-based simulation which enables two agents to maneuver against each other in air-to-air combat. This effort is documented in the following chapters. Chapter II reviews the literature associated with current approaches in AI system architectures. Chapter III

explains the methodology used in designing PDPC, while Chapter IV explains the implementation. Chapter V discusses the obtained results, and Chapter VI closes with the research conclusions. Appendix A briefly discusses some of the mathematics involved. Appendix B provides a listing of the entire rule-base and associated functions.

II. Literature Review

2.1 Introduction

The purpose of this chapter is to review the literature associated with artificial intelligence architectures and their planning methods. I am most concerned with those architectures which can produce real-time performance. Real-time architectures have the following characteristics (7):

- Speed
- Responsiveness
- Timeliness
- Graceful degradation.

Selecting an appropriate planning methodology will ensure that PDPC will exhibit real-time characteristics. Three choices are possible for my planning methodology: take things one step at a time without really trying to plan ahead, create a plan that is likely to succeed, or combine the two extremes to find a suitable compromise. There are numerous approaches presently available to implement a real-time architecture. They extend through the entire range of planning methodologies. The major approaches follow.

Also important is the source of knowledge used in an air-to-air simulation. Several sources were used which accurately describe pilot decisions and maneuvers in air-to-air combat. I use these sources as the domain experts in air-to-air combat.

2.2 Time-Dependent Planning

Time-dependent planning problems are characterized by there being a variety of reactions to predicted events and a range of response times occurring in practice (5). Dean and Boddy propose a theoretical framework for solving time-dependent problems using anytime algorithms, but they have not implemented this approach

for solving real-world problems. One solution is to use an algorithm whose quality of results depends on the amount of computation time. These algorithms, known as anytime algorithms, introduce a tradeoff between computation time and quality of results. The most important characteristics of these algorithms are that (i) they can be suspended and resumed with negligible overhead, (ii) they can be terminated at any time and will return some answer, and (iii) the answers returned improve in some well-behaved manner as a function of time (5). A particular type of anytime algorithms, labeled interruptible, is applicable for thesis effort. Interruptible anytime algorithms produce a result even when interrupted unexpectedly (27). This type of planning does not provide optimal answers, but *good enough* solutions.

2.3 SOAR

Presently there is a team of researchers working on an air-to-air combat simulation using the SOAR architecture. This project shows promise as SOAR is a sophisticated, cognitive architecture that attempts to approximate human intelligence. As the system learns its performance improves. So far they have not developed a knowledge base to the depth which would enable an agent to fly against another (16). SOAR presently is not a practical architecture to use in a real-time system. However, future success is anticipated as the researchers concentrate on more complex behaviors.

2.4 Reactive Systems

Reactive systems avoid planning altogether. Reactive systems use the observable situation as a clue to which one can simply react (25). These systems collect information and use that information to look up a plan to execute next. All the plans are already present in the system. Knowledge about decision making is built into the system *a priori*. A reaction consists of a series of reflexive behaviors, and can display surprisingly complex behavior (1).

An interesting effort in the field of robotics is the use of reactive systems in robots performing space exploration. NASA has developed a standard architecture to use when designing robots for space exploration. This architecture is called the NASA/NBS Standard Reference Model (NASREM) and is hierarchically structured into multiple layers of decreasing complexity and horizontally partitioned into three sections: sensory processing, world modeling, and task decomposition (3, 20). Researchers have used this architecture to develop knowledge-based robotic systems for use in industry (3).

2.4.1 Subsumption Architecture. An architecture is normally composed of a series of vertical layers. Each layer is responsible for more complex behaviors. The first layer obtains the input, manipulates it, and perhaps performs some action. The manipulated information is then passed onto the next layer to support more complex behavior. Brooks proposes a subsumption architecture which consists of a series of horizontal layers (2). Each layer receives input at the same time and can immediately manipulate the input for its purposes. This architecture decomposes in terms of behavior instead of functional modules. This architecture has been successfully implemented on a mobile robot which can navigate around obstacles in a hallway.

There are variations of this architecture presently being worked on. There are researchers who have integrated high-level planning activities with lower level reactive behaviors. This approach is similar to Brooks', since behaviors are defined as fine-grained as possible and are combined to define a behavior. This approach is used to implement the navigational control of a real-time autonomous vehicle control system (24). The layers of complex behaviors are labeled levels of competence. The higher the level of competence, the more complex the behavior.

2.4.2 Embedded Planners. There have been attempts to embed a planner within a reactive system, hoping to obtain the best of both worlds. This integrated

approach attempts to use a planning methodology appropriate to the situation. If the approach is successful, then the disadvantages of the two combined approaches are overcome, and the advantages retained.

Some have attempted to embed a reactive controller in a classical planner-based architecture to enable a system to operate in a dynamic domain (26). The architecture receives input, extracts the initial and goal state, and sends them to the planner. The execution layer can either receive a plan from the planner or respond reactively to the input. This work is theoretical, and has not been applied to a real-world application.

Another approach to building a reactive system is to check if the goal is achievable before constructing a plan. This method extends explanation-based learning to enable general reactive plans to be learned from observation (11). This system can determine if a goal is achievable without having to determine the actions necessary to achieve that goal. There are three classes of problems for which an achievability proof can be constructed: those that consist of repeated actions and a terminating goal, those that consist of constantly changing quantities and intermediate goals, and those that contain multiple opportunities.

Yet another architecture uses a reactive system but when forced to plan uses explanation-based learning to add to the present set of plans. This Theo-Agent architecture can reduce task time from several minutes to under a second once it learns the task (22). This architecture is used by a robot agent which reacts to its surroundings. Once it encounters an unfamiliar situation, the robot agent formulates a stimulus-response rule which it adds to its reactive plan set. This architecture was used to control a Hero 2000 mobile robot to search a laboratory for garbage cans. Presently, the robot operates in a simple domain, and is not capable of complex behavior.

Another similar view is to incorporate adaptability and anticipation into the system. Whitehead and Ballard propose an architecture which uses a look-ahead

mechanism for anticipating better plans (33). This system is adaptable because it estimates the value of internal states and prioritizes the rules it uses to create a plan. They maintain that adaptability is essential to a reactive system since it is unlikely that a designer can anticipate every situation, especially in a dynamic domain.

This adaptable reactive system (ARS) is a parallel, adaptive rule-base system. ARS operates by constantly sensing the state of the world. It uses a bidding mechanism which selects a rule from a set of active rules based on an assigned priority. A utility estimator assigns the priorities by checking the present state of the world. ARS is able to generate new rules and replace old, less useful rules. This system is currently being implemented and tested on simple problems such as stacking blocks.

Kaebling presents a simple architecture which at the top level is composed of a perception and action layer. This approach is similar to Brooks' (2). Kaebling proposes an iterative planning system. The planner works incrementally, storing its state when other parts of the system need to do work (17). This methodology has been implemented on a robot which executes simple tasks such as moving down a hallway without crashing into the wall.

2.4.3 Behavior Control Systems. This architecture uses a reactive planner. Instead of using a large number of primitive actions that are discrete, it implements actions that are continuous. The controller reconfigures the system to use a small amount of primitive actions in different ways instead of relying on a substantial amount of primitive actions. The central concepts of this architecture are (9):

1. Activities form a useful abstraction for discussing and representing primitive actions.
2. Activities can be implemented in a control system as instructions to enable, disable, and configure a fixed set of sensing and action routines.

2.4.4 Universal Plans. The idea of a universal plan is closely tied with reactive systems. It consists of a set of plans which say (28):

If, while achieving a goal, a particular condition occurs, then perform this action.

A universal plan contains a specific reaction for every anticipated situation. If a system using a universal plan encounters a situation for which it has no specified plan, it uses a default plan. A universal plan selects the action to be implemented at execution time. The actions a universal plan can select are designed so that they always move the current situation closer to the goal state in a cooperative world. The difficulty lies in building the contents of the universal plan. The universal plan structure replaces procedural indexing with sensory indexing; makes explicit the conditions under which actions are applicable; renders notions of success and failure irrelevant at execution time; and encourages hierarchy (28).

MAXIM was built using a simple universal plan (8). Its universal plan consisted of three simple behaviors: search, attack, or evade. It switches between plans depending on what the aircraft senses.

NASA's Extra Vehicular Activity (EVA) Retriever robot is currently being developed to retrieve untethered tools and astronauts on the Space Shuttle. EVAR is essentially a free-flying robot. It uses universal planning; Schoppers describes the synthesis of universal plans, intermixing sensing and effecting (29).

2.5 Rule-based Systems

Some researchers have attempted to create real-time systems using a rule-based approach. Since the CLIPS language was created by NASA, it is used for their own expert systems. Efforts are underway to create rule-based systems for the real-time interpretation of space shuttle telemetry data (4).

The CLIPS language uses the Rete net to efficiently perform pattern-matching with many objects and many patterns. The efficiency of the Rete net hinges on the data flow. It depends on the state space not changing frequently to obtain its performance advantage. In this simulation, this cannot be avoided. Researchers have identified the real-time shortcomings of the Rete net and have recommended

and created enhancements (19). These enhancements were taken seriously since the newest version of CLIPS has incorporated these enhancements.

CLIPS is a forward-chaining system, reasoning from data to goals. There have been efforts to implement backward-chaining in CLIPS so that a goal-directed approach can be used when necessary. Homeier discusses an extension of CLIPS called ECLIPS, which is an extended CLIPS for use with goal-directed reasoning (15). It is also possible to implement simple backward-chaining in a forward-chaining system (12).

2.6 Maneuver Information

Shaw has written an exceptional text on air combat maneuvers (31). After an extensive search for texts on air combat, I have found that nothing approaches Shaw's book in terms of completeness and clarity. Most books on air combat consist of personal histories or strategic plans. Shaw covers the basic principles of air combat tactics and maneuvers from a neutral position. He clearly indicates when each maneuver or tactic is appropriate, but remains flexible in their application.

Titan Systems conducted an analysis of pilots' decisions for the Pilot's Associate Program (32). They interviewed pilots from the Air Force, Marines, and Navy. The interviews consisted of mission scenarios. The pilots identified the decisions they made during each phase of a particular mission. The researchers recorded the data and created prioritized tables of information requirements, items, and decisions that are made by a pilot (32). The study is quite valuable to individuals interested in the data flow of air combat.

2.7 Verification

Nazareth has written an article that addresses the issue of rule-base verification. He creates a general taxonomy and outlines the implications for system performance

2.8 Summary

The architectures that hold promise for a reactive simulation are found primarily in the field of robotics. There researchers concentrate on architectures which can continually monitor sensory input and execute an appropriate behavior. I selected a fast, efficient architecture for simulation use based on one of the planning methods reviewed. The architecture selected for this thesis effort is a rule-based, reactive system which utilizes globally available data.

III. Methodology

3.1 Introduction

PDPC is a rule-based reactive system. Captain Hipwell and I use an object-oriented approach in designing and implementing this program. CLIPS rules supply the expertise for the control of the objects. The rules manipulate the objects through associated functions and asserted facts.

Programs which simulate dynamic environments are always difficult to create. Creating a simulation that can realistically perform in a fluid, fast-changing environment such as air-to-air combat is especially challenging. This is due to the following reasons:

- The environment is changing so quickly that it is impossible or not useful to track every changing parameter. However, there are key parameters that must be monitored for proper action selection.
- If an agent executes the wrong action, there is no way to backtrack and execute an alternate action. An incorrect action could be an agent's last action.

3.2 Method Justification

Artificially intelligent programs use some type of planning to execute the best behavior for a particular situation. There are numerous approaches to planning as discussed in the previous chapter. A traditional planning system computes several steps of a problem-solving procedure before executing any of them. This method may become very time consuming, requiring search through potentially large problem spaces. In a dynamic simulation such as PDPC, much of the information required is not available until execution time. Traditional planners also have no mechanism to react to new situations; they might endlessly search a problem space. The central problem with traditional planning systems may be viewed as over-commitment; the systems have strong expectations about the behavior of the environment and make

strong assumptions about the future success of their own actions (10). A reactive system normally does not contain an explicit planner. A reactive system quickly assesses the situation and executes a behavior. A set of simple, reactive behaviors can implement surprisingly complex behavior (25).

We decided on a rule-based approach to create PDPC. Using a rule-based approach enables the encoding of knowledge into easily digestible pieces called rules. The process was incremental, starting with a complete set of facts to define the environment and rules to fire appropriately. Starting with only facts, rules, and functions enabled us to trace in detail the program's data flow. This system evolved into an object-oriented system that still uses facts and rules, with the important players and plans implemented as objects. We essentially used a rule-based approach, global data, and a forward-chaining inference engine to create a reactive system.

The architecture of a rule-based system such as CLIPS parallels the developed maneuver architecture. Each rule in CLIPS is of the form *situation* $\rightarrow$ *action*. This form is almost identical to the basic structure of a simple reactive system.

The initial version of PDPC described the entire world with rules and facts. A number of facts were asserted for each agent, and the number of facts became quite burdensome. Updating information about an agent consisted of retracting and asserting facts. We eventually switched to an object oriented system using COOL. The implementation of COOL reduced the number of facts dramatically because information about an agent was now stored in the slots of an object. Some facts were retained because they were needed for control and rule firing. We also took advantage of the standard OOP characteristics such as inheritance and message-passing. COOL has the ability to pattern match on object slots, so we kept our basic rule-based structure. Switching to the newest version of COOL has reduced memory usage and reduced execution time.

CLIPS has a number of useful features that makes it ideal for quick and efficient implementation of knowledge-based systems. CLIPS combines three major

programming paradigms. CLIPS allows the use of functional, rule-based, or object-oriented style programming. It is also possible to use a combination of the three paradigms. A good example is a rule that pattern-matches an object and calls a function in a conditional element.

CLIPS provides for modularity of rule-bases. CLIPS enables a user to partition encoded rules into modules. Rules are not visible across different modules; this requires some rule duplication. The advantage gained by using modules far outweighs this minor increase in code size.

CLIPS is written in C, and is available with all the source code. This enables the user to modify the source code for his own purposes. Not only can the user modify the source code but also write his own C functions that CLIPS can call.

CLIPS can create a stand-alone executable. A user can create a program in C and embed a knowledge-based program created separately. The C program can seamlessly call the embedded knowledge-based program and provide it input or receive output.

CLIPS is written by NASA's Software Technology Branch. They continue to upgrade the program and use it in their own systems.

We originally considered using LISP as the implementation language for our system. LISP is a powerful language and is traditionally used in artificial intelligence software. In developing CLIPS NASA stated the following constraints with using LISP as a base language for an expert system (6):

- The low availability of LISP on a wide variety of conventional computers.
- The high cost of state-of-the-art LISP tools and hardware.
- The poor integration of LISP with other languages (making embedded applications difficult).

LISP implementations have overcome these disadvantages. but it was the advantages of CLIPS that convinced us to select it as an implementation language.

3.3 Background

A rule-based system is comprised of a collection of rules in the form of:

If this situation occurs, then perform this action

The left-hand side (LHS) of the rule is known as the antecedent, while the right-hand side (RHS) of the rule is known as the consequent. If the conditions of the LHS are satisfied, then the RHS actions are executed. These rules are a natural way to encode knowledge for a particular domain. CLIPS consists of three basic elements: the fact-list, knowledge-base, and inference engine. The fact-list contains the global data. In COOL, the object slots also contain data. The knowledge-base holds all the rules. The inference engine controls overall execution. It decides which rules should be executed and when.

CLIPS is a forward-chaining system. Its reasoning flows forward from the present situation to a selected action. This strategy maps an initial state to a goal in a forward direction. This type of system is well suited for our simulation, since a sub-goal is not established until the present state of the world is examined. If there are many goals, reached by many different paths, then forward chaining may be superior to backward-chaining (18). The ultimate goal of an agent in this simulation is achievable through a sequence of sub-goals. The ultimate goal is to obtain an advantageous position behind a target. For an agent to accomplish this goal, it must accomplish, or fly to, each sub-goal until the ultimate goal is satisfied.

At run time CLIPS checks for facts that are asserted and pattern-matches them with the rules. Any rule that has its LHS side satisfied is placed on the agenda. Rules can be prioritized in CLIPS by assigning a salience. Satisfied rules are placed on the agenda in order of decreasing salience. CLIPS then fires the rules, starting with the first rule on the agenda. The program continues until no more rules can be placed on the agenda. To prevent continual firing of the same rule, CLIPS uses refraction. Refraction limits a rule to firing only once for a particular pattern-match.

I consider PDPC to be a large knowledge-based reactive system. It cannot be considered an expert system because it does not exhibit some of the traits associated with expert systems. It has no explanation facility nor can it improve itself by adding knowledge to itself.

3.4 Development Approach

The basis of a linear model for expert system development was used to develop PDPC. The linear model consists of the following phases:

- Planning
- Knowledge definition
- Knowledge design
- Code and checkout
- Knowledge verification
- System evaluation

During the planning stage a rudimentary work plan was established that served as a guide for development. During this stage the feasibility of the effort was assessed. The design portion of this model is concerned with two major tasks: knowledge definition and knowledge design. The remaining phases of the model are discussed in the latter chapters.

3.4.1 Knowledge Definition. The main objectives of the knowledge definition phase was to define the knowledge requirements of the system. Defining the knowledge requirements consists of identifying and selecting the knowledge sources and knowledge acquisition, analysis, and extraction.

Knowledge source selection and identification are important since the source provides the baseline information for the entire system. The knowledge source acts as the domain expert. For this system, sources were identified and selected according

to importance and availability. Knowledge source availability was considered to be of most importance due to limited time and resources. The majority of maneuver information was extracted from Shaw's book, *Fighter Combat.* Shaw's book describes fighter maneuvers in great detail and concentrates on their employment using a variety of tactics. Another important document was *The Titan Report*, a report sponsored by Wright Laboratories to investigate what information a pilot considered important while conducting a mission.

Once the sources were selected the knowledge acquisition process began. Knowledge acquisition began with reading and studying the relevant documents. The documents provide an extensive amount of knowledge regarding air-combat. It was necessary to identify the specific knowledge that is required by PDPC. This consisted of identifying the basic maneuvers necessary to successfully engage an adversary, what parameters need to be monitored, and what computations need to be calculated.

Once the specific knowledge was extracted, it was necessary to classify and group the knowledge for inclusion into the system. A close examination of maneuvering revealed that there are specific maneuvers appropriate to different situations, or phases in air-combat. This separation of maneuvers helped define the phases designed and discussed in Section 3.4.2.

There is a considerable amount of information that a pilot has to consider when selecting a maneuver. *The Titan Report*, conducted for the Pilot's Associate Program, identified 40 key information items that a pilot must process during air combat (32). There are several key parameters that are considered the most important and greatly influence the maneuver selection. The key parameters used for maneuver construction are:

Target distance Target distance is the three-dimensional distance between two aircraft. This parameter is used extensively to trigger certain maneuvers and phase transitions.

Angle off the tail (AOT) This parameter is an angle that describes the angle between the pilot's line of sight to the enemy aircraft and heading of the enemy aircraft. Reducing this angle is one of the primary goals in maneuvering against an enemy.

Target heading and bearing Target heading is simply the direction the target is flying. Target bearing is the difference between the pilot's heading and the heading of the line of sight to the target. The position and intent of an aircraft influences what maneuver is best used to approach it.

There are basically two types of maneuvering in air combat (31):

Angles The pilot attempts to gain a positional advantage, and then maintains or improves the advantage until firing parameters are achieved.

Energy The pilot attempts to gain an energy advantage while not yielding a positional advantage. Potential energy in this case is defined by this equation:

$$E_s(ft) = H + V^2/2g$$

where H = an altitude above some reference, V = true airspeed (ft/sec), and g = acceleration of gravity (ft/sec^2). The pilot then converts the energy to a lethal positional advantage, without surrendering the entire energy margin.

The agents in this system attempt to gain an angular advantage on their opponents. The angles fighter attempts to gain the angular advantage and thus sets himself up for a missile shot. The energy fighter is more suitable for guns employment since in the energy fight it is usually snapshot opportunities that are acquired. As the program becomes more complex energy maneuvers can be added and an agent can pick between the two.

Since this program performs reactively, I had a greater influence in the maneuver selection than if the program was designed to search a problem space for a maneuver. The maneuver rules were designed so that they knew when to fire and

when not to fire. There are situations where more than one maneuver is applicable. Simply duplicating the LHS of rules can lead to unpredictable behavior, since multiple rules will be placed on the agenda and it is difficult to control which one will fire first. There are times when similar, multiple rules are placed on the agenda, but their firing is controlled by the rule's salience. The higher salience rule will fire first and assert a fact that will clobber the other, similar rules on the agenda.

3.4.2 Knowledge Design. The main objective of the knowledge design phase was to produce a detailed design for this rule-based system.

The acquired knowledge for this system is primarily represented by rules. Agents in the simulation are represented by objects. The objects contain slots that hold information that describe the state of the object at that particular time. Facts are primarily used for control. Their assertion enable the proper rules to fire at the proper time.

Knowledge can be classified as domain-specific or general-problem-solving. If domain-specific, it is either essential or heuristic knowledge. Essential knowledge helps define the search space and provide criteria for determining a solution. Heuristic knowledge can improve efficiency of reasoning by informing procedures of the best place to look for a solution.

The majority of rules in this program contain essential, domain-specific knowledge. This is necessary for a reactive system. This program does not spend time searching for a solution. If the conditions are right for a particular maneuver, then the rule fires and the maneuver is selected. For the maneuvers, there is no separate knowledge base. The necessary knowledge is encoded in the rules that select a maneuver.

The agents in PDPC are designed to be autonomous. They are not explicitly controlled by an external interface. The agents also cannot be explicitly instructed to execute a particular behavior. Observing a particular behavior is accomplished by

adjusting the initial conditions and states of the agents. The simulation is designed to execute from beginning to end with no interruption.

The internal structure of the program depends on modules, fact, and objects. Logical groupings of rules are partitioned into separate constructs called modules. These modules are used to restrict access to rule groups. Most of an agent's state information is stored in object slots. The majority of facts in PDPC are used for internal control. Facts used in PDPC all follow the same format:

(fact side name item item-description)

The first term in each fact is the word *fact.* This term prefaces each fact because CLIPS does not allow the first position in a fact to be occupied by a variable. The remaining terms in the fact structure are variables whose values depend on which agent is active. The *side* slot is occupied by the agent's side, which is either friendly, enemy, or neutral. The *name* slot contains the name of the agent. The *item* slot holds the control item. An example *item* value is MANEUVER. The *item-description* slot contains an atomic description of *item.* An appropriate *item-description* for MANEUVER is LEAD-PURSUIT.

PDPC uses a simple class hierarchy shown in Figure 3.1. At the top of the user-defined class hierarchy is the class PLAYER, which inherits everything from COOL's USER class. Every user-defined class in COOL should inherit directly or indirectly from the class USER, since this class has all the standard system message-handlers attached to it (6). There are two subclasses of PLAYER. PLATFORM is a class defined for moveable objects, while STATION is a class for stationary objects. The PLATFORM class has two subclasses appropriately named MISSILE and AIRCRAFT. Any additional flying objects would be added at this level of the class hierarchy. There are also two additional classes that do not inherit from the PLAYER class. The HISTORY class is used to create a circular queue to track an agent's locations. The PLANS class is used to create and maintain any type of navigational plans, such as a series of waypoints. The PLANS class has three sub-classes, PHASE-PLAN, ROUTE-

PLAN and MANEUVER-PLAN. PHASE-PLAN instances contain a sequence of phases, while ROUTE-PLAN instances contain a route. MANEUVER-PLAN instances will be used by an embedded planner for adjusting maneuver selection. Additionally, the class ROUTE-PLAN has three sub-classes, INTERCEPT-PLAN, LANDING-PLAN, and BINGO-PLAN.

We used descriptive names for the object slots. Numerous slots were created to accurately track the present state of an object. Figure 3.1 lists the slots of each object. They are mostly self-explanatory, and are used to update the state of each object.

3.4.2.1 Phase construction. The basis of the phase construction and architecture was initially developed during a class project for CSCE624, Knowledge Based Systems. The structure of the phases helped to focus the knowledge acquisition effort. Each phase has a particular set of maneuvers and flight characteristics associated with it.

The Titan Report categorized a pilot's decisions into only three phases: pre-engagement, engagement, and disengagement/re-attack (32). However, these decisions concentrated on destroying a target, not on other mission essential tasks such as taking off, landing, etc. Our phase architecture has been designed to enable an agent to follow a logical progression of events.

Figure 3.2 illustrates the phase design. The connected blocks indicate a typical sequence of phases that a pilot would undergo if the mission was flawless. The unconnected blocks clustered around the middle of the diagram indicate the possible phase transitions that could occur, depending on the situation. In the next chapter I discuss the implemented phase transitions.

3.4.2.2 Maneuver Base Construction. Once we decided on the type and number of phases, work began on creating the maneuvers. Each phase normally

has some specific type of maneuvers associated with it. Most of the phases require some type and amount of maneuvering. There are some phases, though, in which the aircraft constantly maneuver at their maximum ability. These are the avoid, engage and evade phases.

A systematic approach was used to create the set of maneuvers. To ensure that each critical situation was covered by a maneuver, a 360° plot was used to ensure that whatever positional relationship exists between the two aircraft, there is a maneuver that an agent can use. Another research effort using discovery-based learning in air combat simulation uses a similar approach (21). There are two situations when there may not be a specific maneuver. The first situation occurs when the knowledge source listed no specific maneuver for that situation. The second situation occurs when a particular situation was not accounted for and initial testing failed to reveal the deficiency. For either situation a default maneuver has been added to fill the gaps. One of the advantages of using a rule-based system is that not every situation must be explicitly encoded. The default maneuver simply implements a pure pursuit and allows graceful degradation in performance.

3.4.2.3 Phase and Maneuver Relationship. Each phase only requires a subset of the total maneuvers available. The only exception is the Engage phase, where the vast majority of maneuver rules are located and required. Pursuit phase consists of rules that enable an aircraft to close in on a targeted aircraft. Engage phase consists of the rules that enable an aircraft to gain an advantageous position behind the targeted aircraft. Evade phase consists of the rules that enable an aircraft to regain an advantageous position if it finds the enemy in its rear quarter. Avoid phase contains the rules that enable an aircraft to avoid a missile. Fire phase has the rules to launch a missile, and Analyze phase has the rules that enable an aircraft to closely follow its target while a missile is tracking it.

3.4.3 Code and Checkout. The main objective here is to encode the design. Actual implementation details are discussed in Chapter IV.

3.4.4 Knowledge Verification. The knowledge verification phase of the linear model involves determining the correctness, completeness, and consistency of the system. These topics are discussed in Chapter V.

3.4.4.1 Verification. Program verification requires extensive review of the rules. Nazareth states that the explicit integration of domain knowledge with verification mechanisms should not be done. However, he does recognize that its incorporation can improve error detection significantly. (23). For PDPC, verification was completed using the integrated domain knowledge. An incremental testing strategy was used to verify the correctness of the knowledge base as the knowledge base developed. The test plan had two major phases: inter-module and intra-module testing. First, the rules within a module were encoded and tested separately to ensure that they fired. Testing was accomplished by asserting a set of initial conditions that favor a certain maneuver. This procedure was followed as each rule was encoded and added to the knowledge base. Once an entire module was tested, the entire module was incorporated with those presently in the knowledge base. Inter-module testing then began, with the assertion of initial conditions that would require phase or module transitions. Extensive use of format statements enabled us to monitor the rule firing. Using the Mac and PC versions of CLIPS also allowed constant monitoring of the fact base and agenda, since those versions of CLIPS allowed simultaneous viewing of the dialog, agenda, focus, globals, instances, and facts windows.

3.4.4.2 Validation. Any system that uses tactics in a fluid environment such as air-to-air combat can be difficult to evaluate. This is due to the rapid flow of information and constantly changing situation. Another important consideration is the subjectiveness involved in applying tactics. However, there are certain,

inviolate fundamentals in tactics that cannot be ignored. The fundamental principles applied in each maneuver phase are as follows:

- Pursuit. Approach the targeted aircraft as quickly as possible. Lead the targeted aircraft.
- Engage. Attempt to gain an angular advantage on the aircraft using whatever maneuver necessary. Attempt to reach and stay within missile firing range.
- Acquire. Once in an angular and range advantage, attempt to stay in that position until a missile is fired.
- Fire. Fire the missile.
- Analyze. Wait until the missile hits or misses the aircraft. Attempt to stay at an advantageous position, but do not try to mimic every move of an aircraft attempting to avoid a missile.
- Evade. If caught in a disadvantage, use whatever maneuver necessary to either regain the advantage or lose the disadvantage.
- Avoid. Attempt to turn within the missile's turn radius, causing the missile to overshoot and miss.
- Breakoff. Leave the immediate vicinity safely.

3.4.5 System Evaluation. The system evaluation phase consists of an overall evaluation of the implemented system and a determination of whether the system's results are correct and desired. This is discussed in Chapter V.

3.5 Summary

This chapter provided the methodology for the development of the design of PDPC and knowledge base. CLIPS is a rule-based language with an internal structure similar to reactive systems. The knowledge sources used for PDPC are Shaw's book titled Fighter Combat, and a technical report commissioned by Wright Labs.

A simple class structure was developed where the main players in the simulation are portrayed by objects. Maneuvers were encoded as rules and the fact list was used for control. All the rules in PDPC are grouped in modules, with each module implemented as one of the many phases in an air combat mission.

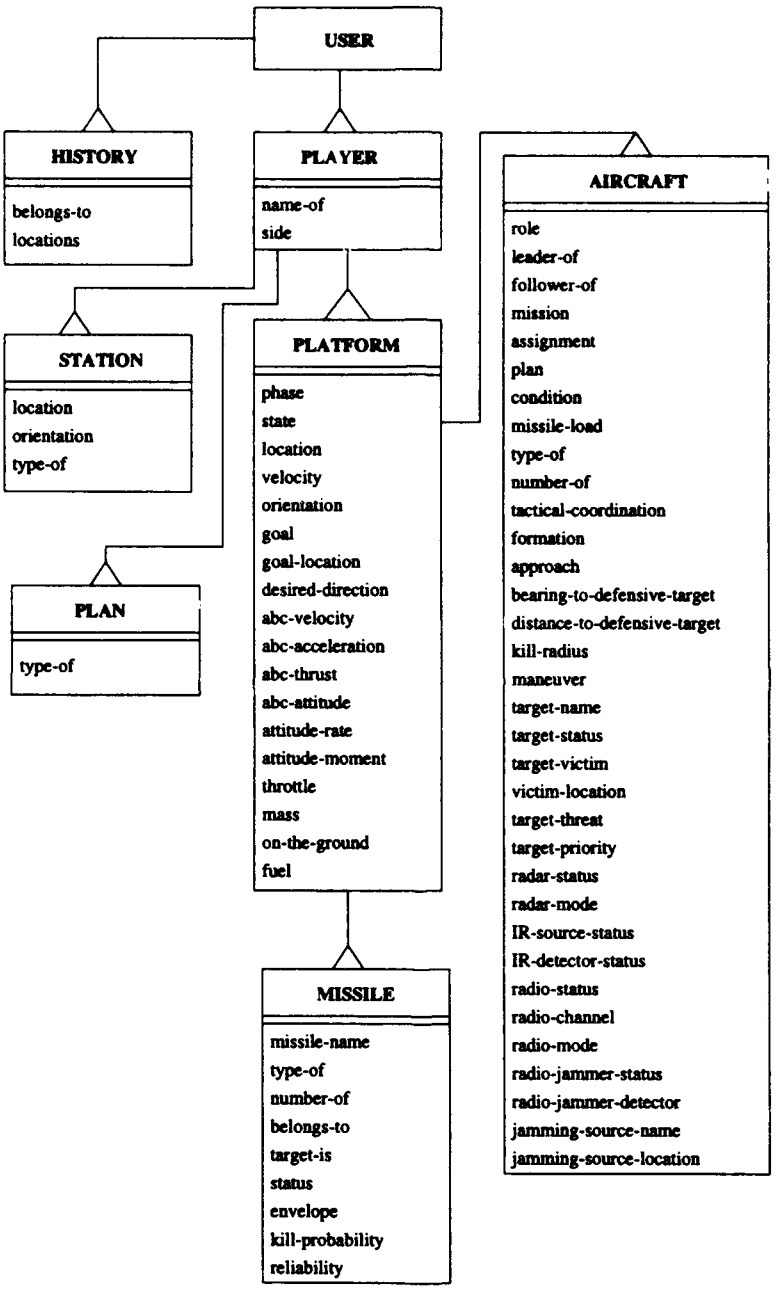


Figure 3.1 **PDPC object diagram.**

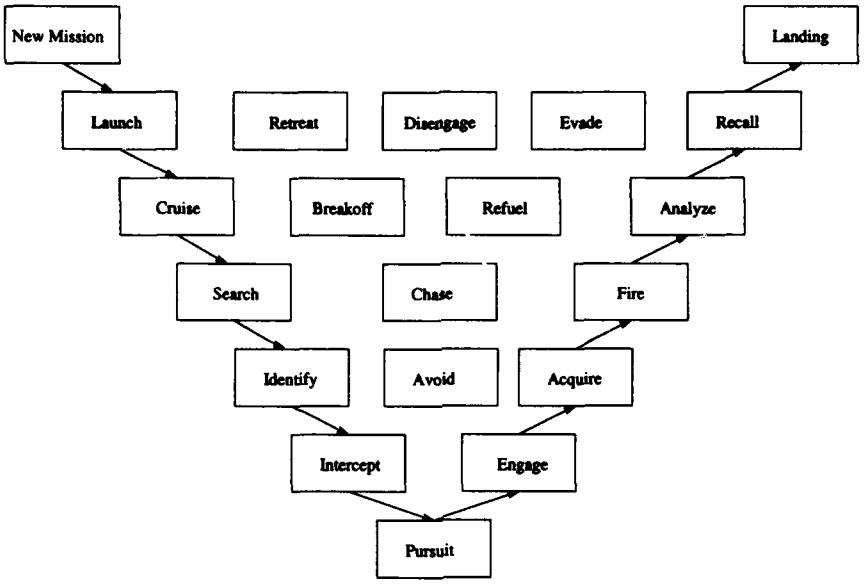


Figure 3.2 **General phase design.**

IV. Program Development

4.1 Introduction

In this chapter I explain the implementation of our design choices. I thoroughly cover the phase and maneuver architecture and explain how maneuvers and their associated phases relate to each other.

4.2 Rule Reasoning Process

The rule reasoning process consisted of two major tasks: creating a phase and maneuver architecture, and implementing the rules required by the architecture.

4.2.1 Phase Architecture. Each phase is partitioned into its own CLIPS module. Most constructs can be exported or imported from other modules except for rules. Rules in one module are not visible to other modules. CLIPS will only attempt to pattern match the rules that are in the current module. This speeds up the program since the inference engine does not have to pattern match all the rules. As an agent switches phases, it uses a different set of rules. This different set of rules is a small subset of the total number of rules in the program.

An agent can move between phases only if there is a rule enabling it to do so. We only have phase-transition rules where required; the majority of the time an agent can only move to a small subset of the phases from any particular phase. Phase transition rules also present an opportunity to trigger other rules or assert facts only one time upon entering a phase. This is particularly useful when entering a phase that requires an instantiation or deletion of an object such as a missile. In the modules containing maneuvers, the phase transition rules have a higher salience than the maneuver selection rules. This ensures that the aircraft is in the proper phase before it executes a maneuver. Not firing the phase transition rules first can

result in maneuver facts applicable in one phase remaining on the fact list while the aircraft is actually in another phase.

At start-up CLIPS defines one module labeled MAIN. The MAIN module contains all the utilities, flight model constructs, and class definitions. Each set of rules for a phase are contained in a module that has the same name as the phase. The *focus* of the program is controlled by using a focus control rule that checks the phase of an agent and changes the focus of the program to that module. If there are rules that can fire, they will. If not, focus returns to the MAIN module.

4.2.2 Phase Description. The following is a description of implemented phases in alphabetical order. Table 4.1 briefly summarizes the phase transitions in this program.

Acquire This is the phase where an aircraft attempts to maintain its positional advantage over several iterations; this enables it to activate, ready, and arm the missile and simulates the time it takes to prepare a missile for firing. During missile preparation the target must remain within the missile envelope. If the target flies out of the missile envelope, missile preparation ceases and the aircraft must reacquire the target. Once the aircraft reacquires the target missile preparation continues from where it was previously halted. Once the missile is armed the aircraft can switch to Fire phase. If the aircraft completely loses its advantageous position, it will switch to Engage phase and attempt to regain an advantageous position. Switching to Engage phase resets missile preparation, forcing the aircraft to prepare a missile from the beginning of the sequence.

Analyze In this phase the aircraft follows the target while the missile flies toward the target. It does not attempt to maintain the target within the a second missile's envelope since the target by now should be violently maneuvering to attempt to avoid the launched missile. The aircraft simply attempts to stay

within the rear hemisphere of the target in case it has to fire another missile. If the missile detonates and destroys the target, then the aircraft can switch to Search phase to look for another target. If the target is not destroyed, the aircraft will switch to either Engage or Pursuit phase to attempt to gain an advantageous position behind the target.

Avoid This phase is similar to Evade phase, except that in this phase the aircraft is attempting to avoid a missile. An aircraft will immediately switch to Avoid phase if an enemy missile is detected in the area, even if it is not the missile's target. While in Avoid phase the aircraft attempts to turn tighter than the missile so that the missile will overshoot. The aircraft will stay in this phase until the missile flies out of range or detonates. From this phase the aircraft can then switch back to Engage phase if it has not been destroyed.

Breakoff In Breakoff phase an outnumbered aircraft attempts to fly away from his opponents. An aircraft enters Breakoff phase when it senses that in the immediate vicinity there are more enemy aircraft than friendly. The aircraft then attempts to fly away from the enemy aircraft. The aircraft can then switch to Engage or Disengage phase, depending on a situation reassessment.

Chase In this phase an aircraft begins to chase a target once it turns and retreats. Chase phase ends when the aircraft approaches the limits of its airspace.

Cruise During Cruise phase aircraft follow a plan of waypoints. Once an aircraft flies a certain route, it can start searching for targets, depending on its mission. Before the start of the simulation a set of waypoint plans are created for the aircraft to follow.

Disengage In this phase an aircraft attempts to become decisively disengaged due to a change in the situation; i.e. is overwhelmed by superior numbers.

Engage In this phase the aircraft maneuvers to obtain a positional advantage over its target. This is where the bulk of maneuvers are used. If the aircraft does

obtain an advantageous position, it can switch to Acquire phase and activate a missile. If the target obtains an advantageous position, the aircraft can switch to Evade phase and defend itself. An aircraft in this phase can also switch to Avoid phase if it senses a missile flying toward its vicinity.

Evade In this phase the aircraft maneuvers to nullify or defeat the pursuer's advantage. The aircraft in this phase can assess the situation and select the appropriate evasive maneuver. An aircraft continues to maneuver in this phase until it loses its disadvantageous position.

Fire In this phase the aircraft fires a missile. An aircraft can enter Fire phase when missile preparation is complete and the target is still within the missile envelope. Once the missile is fired the aircraft switches to Analyze phase to track the missile and the target.

Identify In this phase the aircraft attempts to identify a target and determine whether it is friend or enemy. If the target is an enemy, it becomes the aircraft's goal. The aircraft then switches to Pursuit phase to pursue the target.

Intercept An aircraft uses this phase to intercept a target once it has entered friendly airspace.

Landing An aircraft uses Landing phase to return to home base. The aircraft's goal is changed to home base and the aircraft attempts to land when it reaches the home base.

Launch Once a mission is assigned, an aircraft can launch from its home base. Aircraft continue in Launch phase until they are a certain distance away from the base.

New Mission The New Mission phase assigns an aircraft a mission. Presently the mission choices are: defense, offense, superiority, or escort. The mission selection at this time is not comprehensive. The present mission selection provides four fundamentally different intents for an agent.

Pursuit In this phase the aircraft speeds toward the target as quickly as possible using lead pursuit. Once it reaches engagement range it switches to Engage phase.

Recall In this phase an aircraft has been recalled to home base to be assigned a new mission.

Refuel An aircraft switches to this phase when it detects it needs fuel. An aircraft starts a mission with a predefined fuel level. Each iteration the fuel level is decremented to simulate usage.

Retreat An aircraft uses retreat phase to determine which maneuver will allow exit from an area. It will then return to a safe area while guarding itself against a disadvantage.

Search In Search phase an aircraft flies in a pre-defined pattern in a pre-defined area. While in Search phase the aircraft looks for aircraft. If an enemy aircraft is spotted, the searching aircraft will switch to Identify phase to identify the target. If no enemy aircraft are detected the searching aircraft continues to fly its search pattern, and eventually return to home base.

4.2.3 Maneuver Architecture. I created a hierarchical, vertical architecture for maneuver selection that is composed of three levels. Figure 4.1 illustrates this architecture. The first, or top level uses globally-available data to select a maneuver. The rules then calculate a sub-goal for the agent to fly to, depending on maneuver selection. The goal location for an agent is the location of its target. The sub-goal location is a location that an agent needs to fly to properly position itself behind its target. Each maneuver rules calls a function that calculates a sub-goal for that particular maneuver. Speed adjustment is also calculated at this time. PDPC normally uses range from the sub-goal to adjust speed. The derived information is then sent to the flight model which calculates new position data for the agent. The

Phase	Description	Possible transitions
Acquire	Inside firing envelope, not locked on target	Fire, Engage, Breakoff
Analyze	Track missile. Note results	Acquire, Search, Engage Breakoff
Avoid	Tracked by missile	Engage, Evade, Search
Breakoff	Leave if necessary	Engage, Disengage
Chase	Chase a target	Cruise, Search, Pursuit Engage, Breakoff, Refuel Recall
Cruise	Move aircraft toward assigned goal	Search, Identify, Recall Refuel, Landing
Disengage	Stop engagement	Retreat, Refuel, Recall
Engage	Maneuver to gain advantage	Acquire, Avoid, Evade Pursuit
Evade	At a positional disadvantage	Avoid, Engage
Fire	Locked on target. Fire missile.	Analyze, Breakoff
Identify	Verify friend or foe. Assign bogey as new goal	Pursuit, Refuel, Cruise
Intercept	Intercept a target	Retreat, Recall Chase, Pursuit
Landing	Near home base. Land.	New Mission
Launch	Aircraft takes off	Cruise
New Mission	Assigned mission	Launch
Pursuit	Move aircraft toward bogey	Engage, Evade, Avoid
Recall	Return home	Landing
Refuel	Proceed to refuel station to refuel	Cruise, Identify
Retreat	Overwhelmed. Leave the area.	Refuel
Search	Look for targets	Identify, Refuel, Recall Cruise

Table 4.1 **Possible phase transitions**

design is modular in that an improved flight model can be substituted with little effort.

PDPC presently does not keep track of its previous states. An agent in this simulation does not remember the previous maneuver that it executed. It merely senses the present situation and reacts accordingly. Each maneuver can be called a reflexive response.

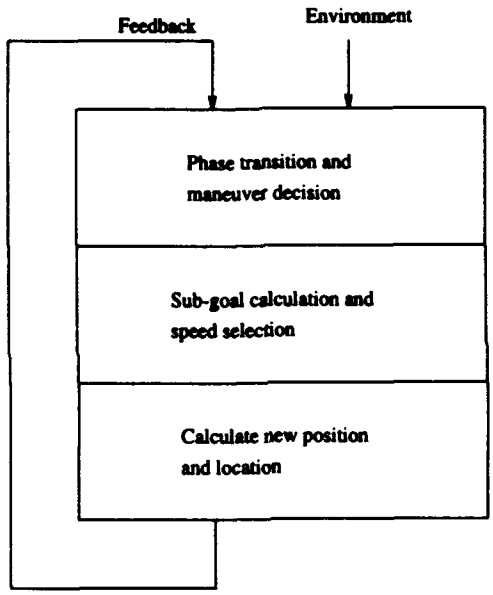


Figure 4.1 **Maneuver architecture.**

The phases designate what rules are available for an aircraft. Each phase can be considered a plan in itself, since each phase should consist of rules for every conceivable situation while in that phase. Each phase also includes rules that will enable an aircraft to switch phases when the situation is appropriate.

The difficulty in implementing the maneuvers is due to determining how to decompose them and how to seamlessly integrate all maneuvers so that the proper

one is always selected. I originally started with the idea of creating a maneuver by sequencing a series of basic, incremental moves such as roll right, pitch up, etc. This method did not prove feasible. Maneuvering in 3-dimensional space requires continuous computations of roll, pitch, and yaw. Rarely is motion in one direction isolated from the other two. Calculating movement in one direction at a time would require a level of granularity that is too computationally intensive. The present implementation creates a sub-goal for an agent to fly to and uses the flight model for orientation calculations.

The method of creating sub-goals for the agent to fly works quite well. In air combat, relative positioning is very important. Everything a maneuvering agent does is in response to the situation. The maneuvering agent keys off its opponent's situation. For every maneuver implemented a point to fly to is calculated that will yield the desired outcome of that maneuver. While executing a lead pursuit, for example, the agent points his nose at a point ahead of its target. Figure 4.2 illustrates the sub-goal location for a lead pursuit. For this pursuit a sub-goal is calculated that lies ahead of the target along its nose vector. The velocity of the target is used to determine the offset along the nose vector.

In some situations, a calculated sub-goal may lie beyond the maximum turn rate of the agent. For example, if a sub-goal is calculated that is behind the agent some distance away, the agent will take a very long time time to reach it. It is not important that the agent will never actually reach that sub-goal. It is important, though, that a certain behavior is induced. In an effort to reach that sub-goal the agent will attempt as tight of a turn as possible. That tight turn is the desired behavior for the agent to exhibit. A few iterations later the agent may have a sub-goal that is located in a completely different direction. It will again attempt to reach that sub-goal by immediately turning about as hard as possible. This type of behavior is analogous to an actual pilot flying. When maneuvering against an opponent he is flying at the limits of his performance envelope.

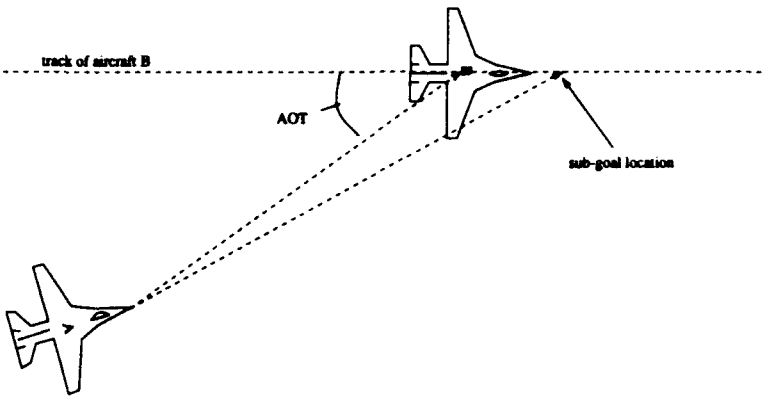


Figure 4.2 **Sub-goal determination.**

The speed control rules use the range to the target to either decrease, maintain, or increase an agent's speed. A pursuing agent will attempt to maneuver into a position where it can effectively deploy its weapon system. In PDPC this position is defined by the minimum and maximum missile ranges. If the agent is within that range envelope, it maintains its speed. If the agent is outside of the envelope, the speed is increased. If the agent is inside the envelope, the speed is decreased. The speed control rules are fundamentally simple. After a maneuver is selected a speed control rule fires and asserts a fact that contains the amount of forward thrust selected. To increase speed, a positive forward thrust is selected. To decrease speed, a negative forward thrust is selected. To maintain speed, zero forward thrust is selected.

Placing all the rules at the top level of the maneuver architecture allows an agent to move immediately from one maneuver to another. The maneuvers are essentially meshed together. In air combat most maneuvers are transient; a pilot executes one immediately after another. The only problem with this is that as more maneuvers are added, CLIPS has to pattern-match against more rules, which increases execution time. This is noticeable when observing the execution of a sample run.

Program execution time while in a particular phase depends on the number of rules in that phase/module. This is especially noticeable between Pursuit phase, which contains only one maneuver, and Engage phase, which contains every maneuver available.

It became clear after studying maneuvers that the more complex out-of-plane maneuvers are actually composed of several simple maneuvers. Out-of-plane maneuvers are those maneuvers that include using a vertical component. Almost every out-of-plane maneuver consists of trading airspeed for altitude and then using lag, pure, or lead pursuit to close in on the enemy. This simplifies complex maneuver construction. A rule can be created that initially detects the conditions for a complex maneuver. This rule will fire when the initial conditions are appropriate and not fire during the latter phase of the maneuver where another rule implementing a simple maneuver will fire to complete the maneuver.

4.2.3.1 Simple Maneuvers. Simple maneuvers are those that are normally executed within the same plane as the pursuing agent and target. Each of these maneuvers are independent and have no assigned sequence. Figure 4.3 depicts the three different type of pursuit. Figure 4.4 shows when each offensive maneuver is appropriate. The aircraft in the middle of the circle is the target. The AOT ranges indicated by the arrows show when an aircraft can execute a particular maneuver with that AOT.

AOT is used extensively in describing when a maneuver is appropriate for a given situation. For clarity and convenience AOT can be divided into three categories: low, medium, or high. Low AOT ranges from 0° to 29°, medium AOT ranges from 30° to 60°, and high AOT ranges from 61° to 180° (31).

Lag pursuit An aircraft can use lag pursuit to decrease AOT while simultaneously maintaining separation. Lag pursuit occurs when the pursuing aircraft points its nose behind the fleeing target. Figure 4.3 illustrates where an aircraft should

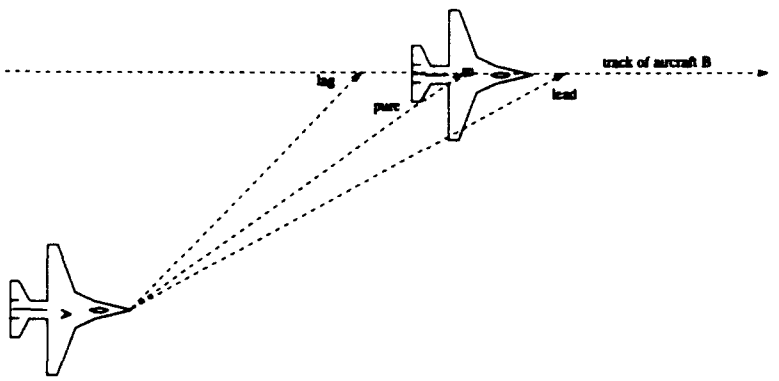


Figure 4.3 **Types of pursuit.**

point its nose for lag pursuit. If the aircraft is in range to fire a missile, but needs to decrease AOT, then it will use lag pursuit. The aircraft's sub-goal location is created behind the target, scaled by the target's velocity.

Pure pursuit Pure pursuit is a pursuit option where an aircraft flies straight toward its target. The target's location becomes the aircraft's sub-goal. Our missiles use pure pursuit in tracking a target. I also chose this maneuver as the default maneuver in situations where no other rule fires.

Lead pursuit Lead pursuit is used when an aircraft needs to increase closure as quickly as possible. In lead pursuit the aircraft points its nose ahead of its target. This maneuver could result in greater AOT. This maneuver is used by all aircraft in Pursuit phase. The aircraft's sub-goal location is ahead of the target, scaled by the target's velocity.

Pursuit variations To maintain a reactive structure, there are several rules implemented where they check for a particular situation, but still implement a previously discussed pursuit. These rules concern themselves with side approaches or when the aircraft are flying parallel to each other. In both cases the aircraft would use a lag pursuit.

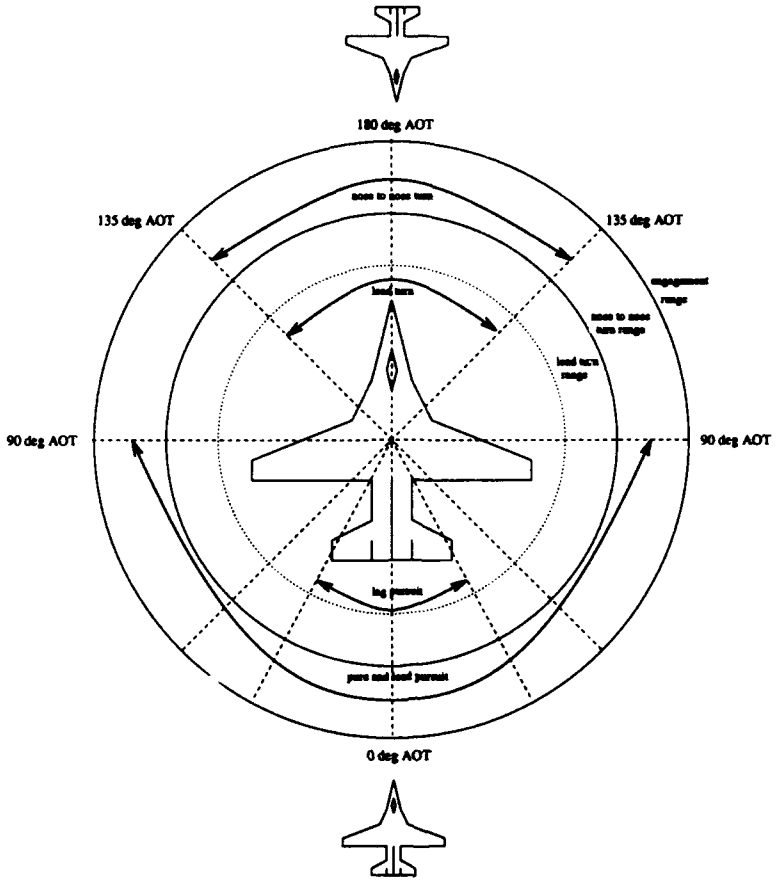


Figure 4.4 **Simple maneuvers.**

Pursuit with separation If separation exists, and the aircraft are approaching each other, then an aircraft will attempt to maintain that separation so that as it passes the other aircraft it will turn and attempt to get behind its target.

Lead turn A lead turn is a maneuver used by an aircraft approaching a target in its forward quarter. The aircraft should have similar speeds, and separation is not necessary. An aircraft attempts a lead turn to obtain a position in its target rear hemisphere. The aircraft's sub-goal location is 45° from the target's heading, scaled by the target's velocity.

Nose to nose turn A nose to nose turn is similar to a lead turn except that the aircraft turn as they pass each other. The aircraft's sub-goal location is established at a right angle from the target's heading, scaled by the velocity.

4.2.3.2 Complex Maneuvers. These maneuvers are labeled complex because they require an out-of-plane move. They also use a sequence of maneuvers for successful execution. A complex maneuver rule establishes the beginning of the maneuver, and is completed by the execution of simple maneuver.

Lag displacement roll This roll is an out-of-plane maneuver. This maneuver is used to reduce ACT and increase range, possibly to meet the minimum range constraint for missile deployment. When the aircraft approaches its target with high speed and low AOT, an overshoot is possible. The aircraft climbs to reduce airspeed and then swoops down on its target. The sub-goal is established on the inside of the turn with an increased vertical component.

High yo-yo A high yo-yo is an out-of-plane maneuver similar to the lag displacement roll. It is useful in preventing overshoot when the aircraft's AOT is medium. If the pursuing aircraft is rapidly closing on its target and turning in the same plane, it can climb and pull up out of the plane. The climb reduces the pursuing aircraft's speed component, thus reducing closure. Once closure is reduced to zero the pursuing aircraft can switch to a type of pursuit to complete the maneuver. The aircraft's sub-goal is established at an altitude difference that substitutes altitude for airspeed. The sub-goal is also established on the inside of the target's turn.

Low yo-yo The low y-yo is also an out-of-plane maneuver that is used to increase closure and angular advantage. It is useful in a situation where a pursuing aircraft does not have the turn capability to pull up and fire at its target without slowing down excessively. To prevent the excessive speed loss, the pursuing aircraft should turn nose down toward the inside of the turn. This

allows the pursuing aircraft to position its nose ahead of the target. Once the pursuing aircraft obtains excess lead it can then level off, climb, and intercept the target. This maneuver is not yet implemented.

4.2.3.3 Defensive Maneuvers. I have implemented a series of defensive maneuvers that an agent can use when evading a pursuing opponent. A defensive maneuver is selected depending on where the pursuing opponent is pointing its nose. To determine where the opponent's nose is pointing the rules compare the opponent's target bearing with its track crossing angle. If the target bearing is greater than the track crossing angle, then the pursuing opponent is using lead pursuit. If the target bearing is less than the track crossing angle, then the pursuing opponent is using lag pursuit. If the target bearing is equal to the track crossing angle, then the pursuing opponent is using pure pursuit. This is one area where perfect information allows the aircraft to act correctly. In the real world, a pilot would not easily be able to determine what kind of pursuit his opponent is using. Figure 4.5 shows when an aircraft would transition to a defensive maneuver.

The philosophy for missile defense is quite simple, and consists of two tenets: (i) prevent the missile from being launched at all, and (ii) failing the first, attempt to present the shooter with the least favorable shot and endeavor to make the missile's task as difficult as possible (31). An aircraft in PDPC selects only one type of maneuver when it detects an incoming missile. It attempts to turn tighter than the missile so that the missile overshoots. This is an adequate maneuver since the missile flies much faster than an aircraft and does not have a large wing surface. Its turn radius, then, is much larger than that of an aircraft, and cannot react as quickly to match the aircraft's turn.

It seems intuitive to most that if an aircraft wants to turn tighter than a missile, it should decrease its speed, since aircraft have tighter turn radii at lower speeds. This is true, but not appropriate for missile defense. Moving at a higher

rate of speed makes it more difficult for a pursuer to acquire. Also, in the case of rear-quarter missiles, having a speed advantage can reduce the target range of the missile by up to 25% (31).

Defense against lag pursuit There are two defensive maneuvers appropriate in this situation. There are two situations to check for, a hot side or cold side lag. A hot side lag occurs when the pursuer is on the inside of an evader's turn. A cold side lag occurs when the pursuer is on the outside of an evader's turn. If it is a cold side lag, then the evading aircraft needs to reverse its turn. The sub-goal is established ahead of opponent. If it is a hot side lag, then the evading aircraft attempts to tighten its turn. The sub-goal is established at the opponent's location.

Defense against pure and lead pursuit In defending against a pure pursuit, the aircraft has three options. If the aircraft is faster than its opponent, then it turns slightly and speeds away. The sub-goal is established ahead of the opponent. If the opponent's AOT is very small then the evading aircraft turns harder so as eliminate a missile shot. If the evading aircraft is slower than its opponent, then it attempts a tight turn toward the opponent. The sub-goal is established at the opponent's location.

Defense against missile If an aircraft detects an oncoming missile then it immediately attempts a tight turn. The sub-goal is established at a right angle from the missile, so that no matter how hard the missile turns the aircraft attempts to turn tighter.

Typically, a maneuvering agent would follow this ideal phase transition sequence:

- An agent identifies a target, enters Pursuit phase, and uses lead pursuit to close the distance to the target as quickly as possible.
- The agent, after closing to engagement distance, switches to Engage phase.

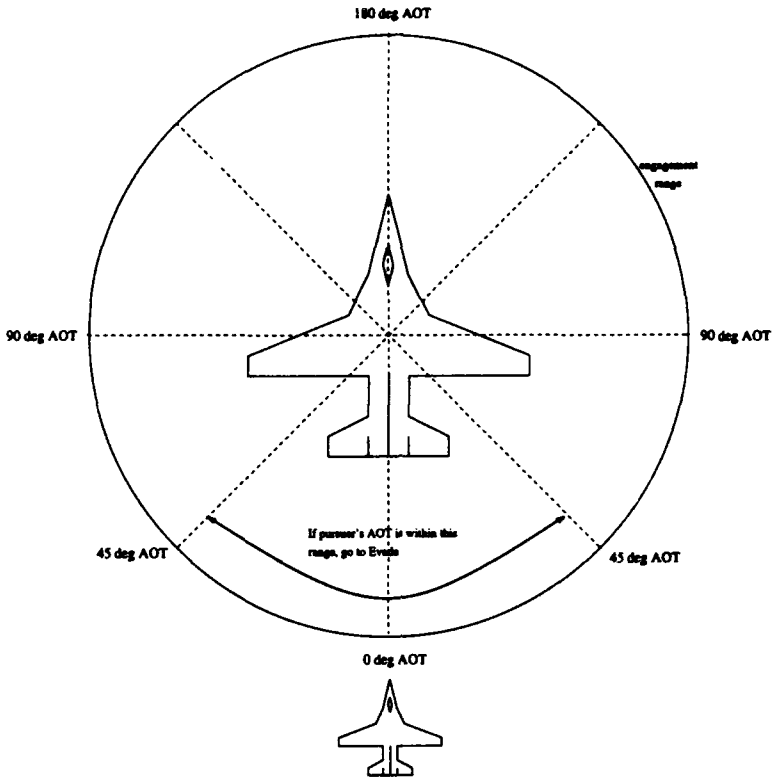


Figure 4.5 **When to evade.**

- In Engage phase the agent maneuvers to get within missile range and behind its opponent. The agent switches between appropriate maneuvers.
- Once the agent obtains a favorable position, he switches to Acquire phase, simultaneously arming a missile. The agent employs a type of pursuit to stay behind the target.
- In Acquire phase the agent prepares the missile for firing. Once the missile is ready, the agent moves to Fire phase and fires the missile. The agent then immediately switches to Analyze phase.

- In Analyze phase the agent still maneuvers behind its opponent, waiting for the missile to detonate.
- If the missile detonates and destroys the target, the agent will switch to Search phase to find another target. If the missile fails to destroy the target, the agent will fire another missile if still within firing parameters.

4.2.4 Flight Model Adjustment. During the early stages of development the agents in this program exhibited large altitude fluctuations. This is due to the altitude component of an agent's sub-goal. The agents will chase each other by diving or climbing, depending on the pitch of the target. There is a rule that adjusts an agent's sub-goal to stay within an optimum altitude range. The optimum altitude range is within 25,000 to 35,000 ft, where a jet fighter can best maneuver (31).

It became apparent after working with several phases that the flight model needed expansion. An agent's behavior is defined by its phase, maneuver selection, and flight characteristics. An agent should exhibit flight characteristics that accurately reflect the nature of the agent's phase. This is accomplished by adjusting the orientation rates of an agent depending on its phase. The orientation rate adjustments are summarized in Table 4.2. All the adjusted rates use the Cruise rate values as a standard rate. Adjusting the Cruise rates will proportionately adjust the rates in the other phases.

I tailored the flight characteristics during the maneuver phases by adjusting the flight model to call different functions depending on the aircraft's phase. The operator rule calculates the orientation of the aircraft before calling on the ruleset to fly it. It calculates a new roll, pitch, and yaw for an aircraft. The rates for calculating the orientation differs depending on what the aircraft is doing. For instance, an aircraft cruising will not have to maneuver as severely as an aircraft avoiding a missile. The tendency is for the aircraft to fly in a more relaxed manner when not engaged, and much more aggressively and violently when maneuvering.

Phase	Roll	Pitch	Yaw
Cruise	standard	standard	standard
Analyze	1	1	1
Avoid	3	0.1	1
Engage	3	1	1
Evade	3	1	1
Fire	1	1	1
Pursuit	1	1	1

Table 4.2 **Orientation rates adjustment**

The orientation limits are also adjusted according to the phase. Most of the phases have limits of 90° for total roll, pitch, or yaw.

It is important to point out that the orientation rates are adjusted due to the agent's behavior, and not due to the aircraft's capabilities. This means that when an orientation rate is increased, it is analogous to a virtual pilot moving the stick in the cockpit with more vigor.

4.2.5 Missile Maneuvers. Today's missiles can implement several different types of pursuit. These pursuits are lead collision, lead pursuit, and pure pursuit (31). Lead collision is the most effective and has the shortest trajectory. But it is also the most difficult to implement and uses a complex navigational system. There are really no advantages to using lead pursuit, since it adds the complexity of lead collision with little advantage. Pure pursuit is a simple pursuit to implement, both in actual missiles and in this program. It has the longest trajectory, but this works well with rear quarter missiles, since a pure pursuit trajectory always approaches the target from the tail.

Missiles in PDPC use simple pure pursuit to track their targets. They use the same speed control, with an increased acceleration rate and higher top speed. In

PDPC a missile's top speed is twice that of an aircraft, while acceleration is 5 times as large.

4.2.6 Planning. The logical extension for PDPC is to add a planner, since a planner has not been implemented. The primary goal was to create a reactive system. There are times when PDPC will have time to choose between several courses of action. PDPC, when faced with a choice in an air-to-air engagement, can use the history of the opposing agent to determine the opposing agent's tendency to maneuver in a particular fashion. PDPC should be able to choose during the following times:

- When an agent has an overwhelming advantage over its opponent.
- When not decisively engaged.
- Executing non-critical tasks.

PDPC has the ability to maintain a history of all agents. From the opposing agent's history PDPC can establish an opponent's tendencies, and use that to its advantage. During an air engagement a pilot does not have time to formulate complete plans, but he can use what he knows about the enemy to his advantage. For example, if a pilot realizes that every time he is on the tail of his opponent, the opponent turns sharply left, then next time that situation occurs the pilot can use that information to predict that action. The pilot can then prepare himself to match the opponent's left turn.

During maneuver selection an embedded planner could influence or suppress the maneuvers that are reactively selected. There are three different decision types that an embedded planner could make to aid in this maneuver selection.

- Select the same maneuver, but with a different sub-goal calculation.
- Select one of several applicable maneuvers.
- Select a maneuver which normally would not be selected.

Presently there is only one way to calculate a sub-goal for a particular maneuver. This limits an agent to the same approach to a target as long as that maneuver is selected. If the agent wanted to execute a selected maneuver, yet approach the target from a different direction, then a planner would need to create a maneuver plan. Waypoint plans are already used in Cruise phase to provide an agent with a route to follow. A planner could dynamically fill the slots of a pre-instantiated maneuver plan. The agent could read this maneuver plan and follow it to approach a target from a different direction. Simple implementation of this type of planning would require the addition of another phase. The agent would then enter this phase when it is required to follow a maneuver plan.

As the maneuver knowledge base increases in size and scope, a situation may occur where several maneuvers are applicable in a situation. A planner could select a desired maneuver by dynamically adjusting the salience of the desired maneuver. Dynamically adjusting the salience of the desired maneuver would ensure that it is placed on top of the agenda, and will fire first. As soon as it fires, the other applicable maneuvers would be clobbered due to the assertion of a control fact by the fired rule.

There may also be times when the desired maneuver is not a maneuver that would normally be selected, yet the planner has decided that the desired maneuver is required in a particular situation. This can also be accomplished by dynamically adjusting the salience of the desired maneuver.

There are numerous areas where an embedded planner could add to PDPC's capabilities. Besides maneuver selection, an embedded planner could be used to determine targets, select weapons, or modify tactics. Captain Hipwell's work addresses some of these areas with his rulesets (13).

4.3 Program Flow

The program flow is illustrated in Figure 4.6. This diagram assumes that the simulation has already started; the agents are flying and have a mission. An agent selects a maneuver depending on the phase it is presently in. Selecting a maneuver automatically calculates a sub-goal and amount of forward thrust to use. The status slot of the object class Platform is used to maintain the status of an agent. The agent's status cycles through these stages as shown in Figure 4.7.

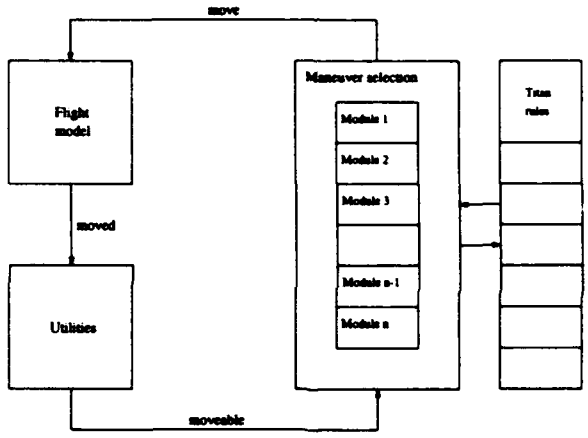


Figure 4.6 **PDPC program flow.**

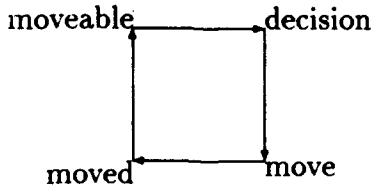


Figure 4.7 **Agent status stages**

An agent's status is *moveable* when it is able to fly. Once a maneuver is selected the agent's status changes to *decision*. It is changed to *move* once the orientation

has been calculated and the agent is ready for a new calculated location. The flight model calculates a new location for the agent. The *move-player* rule, which does most of the work, has the highest salience in the program. This ensures that it will always fire first when placed on the agenda.

The rule calculating the new location will change an agent's status to *moved.* The RHS of the rule that calculates a new location will assert a fact indicating everybody has moved after the last agent has moved. This is accomplished by using a COOL specific query that checks all instances' status for *move.* Asserting this fact triggers the rules in the utility set.

The utility set contains the rules that handle output, decrement the iterator, and update the histories of each agent. A fact is asserted in the beginning of the program run indicating how many iterations the program should execute. This fact is decremented each decision cycle. The locations of all instantiated objects are written to an output file for review. Each instantiated object has a history object associated with it. The history object maintains the last five locations of an agent. This information is used to predict an agent's new location when calculating a sub-goal for lead-type maneuvers. At this time all of the agents' status is changed to *moveable.* The focus-control is also activated at this time. For each existing agent the focus-control checks its phase, and changes the focus to that module. This enables multiple agents to be in different phases.

At appropriate times the aircraft will check the titan ruleset for engagement decisions. The titan ruleset, created by Captain Hipwell, consists of the following modules:

- Pre-engagement
- Engagement strategy
- Intercept geometry
- Counter-action

- Weapons employment
- Post-engagement

The agents, while in other phases, will check a particular titan module, depending on its present situation. The pre-engagement module's function is threat assessment and commitment considerations. The engagement strategy module decides on cooperative behavior between two agents while flying in formation and approaching a target. The intercept geometry module calculates a intercept trajectory for agents. The counter-action module is concerned with the employment of chaff and flares. The weapons employment module handles missile activation and deactivation. Finally, the post-engagement module monitors phase transitions depending on the results of an engagement. For a more detailed explanation of these modules, see Hipwell (13).

4.3.1 Decision Cycle. Program execution in PDPC consists of multiple iterations. Each iteration PDPC executes a decision cycle where a maneuver is selected, a sub-goal calculated, and a forward thrust calculated. Since this is a reactive system, the decision cycle is very short. The following code excerpt contains the rules that decide on a lead-turn maneuver.

The rule *decide-on-lead-turn* is shown in Figure 4.8. The left hand side of the rule *decide-on-lead-turn* describes the conditions present for this rule to fire. The first pattern is a conditional element that checks to see if another maneuver rule has fired. This guarantees the selection of a single maneuver since if more than one maneuver is activated and placed on the agenda, the first one that fires will clobber the remaining rules on the agenda. The rule then pattern matches existing objects of class AIRCRAFT for two aircraft objects that meet the field constraints. The *pursuit-possible* function checks to see if if the aircraft can pursue each other. This function returns a positive value if pursuit is possible, and a negative value if pursuit is not possible.

The AOT is then checked for the aircraft. If it falls within the test parameters, then the conditional element passes. For a lead turn, the AOT should fall between π and $\pi/2$. The speeds of the aircraft are then checked. For a lead turn, the speed difference should not be very large, or the executing aircraft will overshoot its target. Finally, the range is checked. If the range falls within the given parameters, then the LHS is satisfied.

The RHS of the rule now executes. The status slot of the aircraft is updated to *decision*. The maneuver slot is updated with the implemented maneuver. The RHS also asserts a fact that will fire the next rule, *decide-on-lead-turn-point*.

The next rule, *decide-on-lead-turn-point* is shown in Figure 4.9. It fires when the previous rule asserted the maneuver fact. The left hand side pattern-matches the appropriate information and calculates a sub-goal for the aircraft to fly to. It was difficult to determine how to calculate the sub-goal for this maneuver. Experimentation resulted in a sub-goal calculation of 45 degrees off the target's heading. The RHS of the rule updates two slots of the aircraft. The sub-goal is placed in the goal-location slot and the desired-direction slot is updated.

The last rule, *nose-turn-speed-same*, is shown in Figure 4.10. This rule determines the magnitude of forward thrust of the aircraft. In this situation, the aircraft attempts to maintain the same speed while executing this maneuver. For other maneuvers there are rules that decrease or increase the speed depending on the range from the target. Notice that this rule also fires for a nose-to-nose turn; hence the name. The RHS asserts a speed control fact that is used by the flight model.

Once the maneuver, sub-goal, and speed are selected the flight model calculates the orientation angles and new position of the aircraft. The utilities take care of the overhead, and the next decision cycle starts.

4.4 Summary

In this chapter I explained the implementation details of the phase and maneuver architectures. 20 different phases were implemented in PDPC. The following maneuvers were described and implemented:

- lag, pure, lead pursuit and variations
- lead turn
- nose-to-nose turn
- lag roll
- high yo-yo
- avoidance and evasive maneuvers

An agent in this simulation executes these maneuvers by flying to a calculated sub-goal. The program flow was discussed, and a sample decision cycle was stepped through by tracing through the rules for a lead turn.

```
(defrule decide-on-lead-turn
  (not (fact ?side ?attacker maneuver ?))
  ?aircraft <- (object (is-a AIRCRAFT)
                       (phase Engage)
                       (state moveable)
                       (side ?side)
                       (name-of ?attacker)
                       (location ?x0 ?y0 ?z0)
                       (velocity ?dx0 ?dy0 ?dz0)
                       (goal ?defender))
  (object (is-a AIRCRAFT)
          (name-of ?defender)
          (location ?x1 ?y1 ?z1)
          (velocity ?dx1 ?dy1 ?dz1))
;;approaching each other ?
  (test (< 0 (pursuit-possible ?dx0 ?dy0 ?dz0 ?dx1 ?dy1 ?dz1
                               ?x0 ?y0 ?z0 ?x1 ?y1 ?z1)))
  (test (< 0 (pursuit-possible ?dx1 ?dy1 ?dz1 ?dx0 ?dy0 ?dz0
                               ?x1 ?y1 ?z1 ?x0 ?y0 ?z0)))
  (test (>= (pi)
            (angle-off-the-tail ?dx1 ?dy1 ?dz1
                                (line-of-sight ?x0 ?y0 ?z0 ?x1 ?y1 ?z1))
            ?*engage-half-pi*))
;;speeds close ?
  (test (> 4 (abs (- (speed ?dx0 ?dy0 ?dz0) (speed ?dx1 ?dy1 ?dz1)))))
;;In range ?
  (test (>= (* 2 (speed (closing-velocity ?dx0 ?dy0 ?dz0
                                          ?dx1 ?dy1 ?dz1)))
  (target-distance ?x0 ?y0 ?z0 ?x1 ?y1 ?z1)))
  =>
  (send ?aircraft put-state decision)
  (send ?aircraft put-maneuver lead-turn)
  (assert (fact ?side ?attacker maneuver lead-turn))
  (format ?*Engage* "%s will attempt to lead-turn %s%n"
  ?attacker ?defender)
  )
```

Figure 4.8 **Decide on lead turn rule.**

```
(defrule decide-on-lead-turn-point
  (fact ?side ?attacker maneuver lead-turn)
  ?aircraft <- (object (is-a AIRCRAFT)
                       (phase Engage)
                       (side ?side)
                       (name-of ?attacker)
                       (location ?x0 ?y0 ?z0)
                       (goal ?defender))
  (object (is-a AIRCRAFT)
          (name-of ?defender)
          (location ?x1 ?y1 ?z1)
          (velocity ?dx1 ?dy1 ?dz1))
  =>
  (bind $?lead-turn-points
        (create-lead-turn-point ?dx1 ?dy1 ?dz1 ?x1 ?y1 ?z1))
  (assert (fact ?side ?attacker sub-goal
          =(nth 1 $?lead-turn-points)
          =(nth 2 $?lead-turn-points)
          =(nth 3 $?lead-turn-points)))
  (send ?aircraft put-goal-location ?x1 ?y1 ?z1)
  (send ?aircraft put-desired-direction toward)
  (format ?*Engage* "%s has a sub-goal%n" ?attacker)
  )
```

Figure 4.9 **Decide on lead turn point rule.**

```
(defrule nose-turn-speed-same
  (fact ?side ?attacker maneuver nose-to-nose-turn|lead-turn)
  ?aircraft <- (object (is-a AIRCRAFT)
                       (phase Engage)
                       (side ?side)
                       (name-of ?attacker)
                       (location ?x0 ?y0 ?z0)
                       (velocity ?dx0 ?dy0 ?dz0)
                       (abc-acceleration ?dU ?dV ?dW)
                       (goal ?defender))
  (object (is-a AIRCRAFT)
          (name-of ?defender)
          (location ?x1 ?y1 ?z1)
          (velocity ?dx1 ?dy1 ?dz1))
  =>
  (bind ?goal-speed (speed ?dx1 ?dy1 ?dz1))
  (assert (fact ?side ?attacker speed same
                =(speed-control ?dx0 ?dy0 ?dz0 ?dU ?goal-speed)))
  (format ?*troubleshooting* "%s matching speed%n" ?attacker)
  )
```

Figure 4.10 **Nose turn speed same rule.**

V. *Experimentation and Results*

5.1 Introduction

In this chapter I examine several plots of different scenarios that illustrate PDPC's execution. The plots presented here will show that PDPC agents properly behave as maneuvering aircraft. All of the plots show agents already in flight. The agents do not start from New Mission phase, but from an appropriate phase where they begin behaving in a particular manner. For the purposes of brevity and clarity, I shall refer to the friendly agent as *pilot* and the enemy agent as *bogey1*.

5.2 Maneuver Example

Figure 5.1 shows a scenario where *pilot* and *bogey1* are approaching each other. *Pilot* is approaching from the west and *bogey1* is approaching from the east. They both start off in Pursuit phase and switch to Engage phase when they reach engagement range. As they approach each other, *bogey1* is the first to execute a lead turn. *Pilot* reacts by attempting to pursue *bogey1*. However, *bogey1* maneuvers to an advantageous position behind *pilot* and switches to Acquire phase. *Pilot* determines he is at a disadvantage and switches to Evade phase and speeds away from *bogey1*. He then attempts to turn around and re-engage *bogey1*.

5.3 Missile Engagement

Figure 5.2 shows a scenario where the *pilot* is flying behind *bogey1*. Both of the aircraft are oriented toward the east. The *pilot*, who started in Engage phase, switched to Acquire since *bogey1* was within the missile envelope. *Bogey1*, who determined it was at a disadvantage, switched to Evade phase. After three iterations the *pilot* fired a missile and switched to Analyze phase. When *bogey1* discovered a missile in the area, it immediately switched to Avoid phase. The bogey then started to turn as sharply as possible, trying to turn within the missile. While the missile was

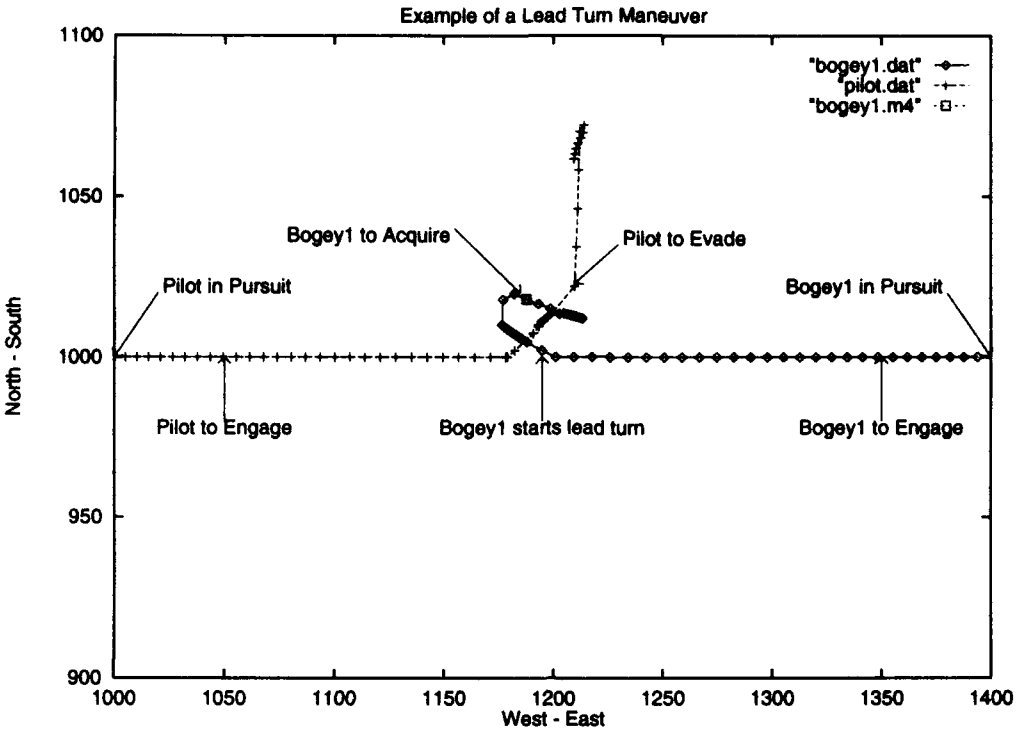


Figure 5.1 **Lead turn execution.**

tracking the *pilot* flew behind and maintained sight of *bogey1*. The missile reached *bogey1* and detonated, but did not destroy *bogey1*. The missile did not destroy *bogey1* because its kill probability is not 100%. Occasionally, a missile will not destroy an agent when it detonates.

After the first missile exploded, the *pilot* switched back to Acquire since hit was still in a favorable position. It then fired a second missile. *Bogey1* immediately saw the second missile and continued to turn sharply in an attempt to avoid the missile. From the figure it does not appear that *bogey1* is turning sharply, but it is maneuvering at its maximum ability, which does enable it to avoid the oncoming missile. This time *bogey1* was successful in its maneuver and the missile overshot.

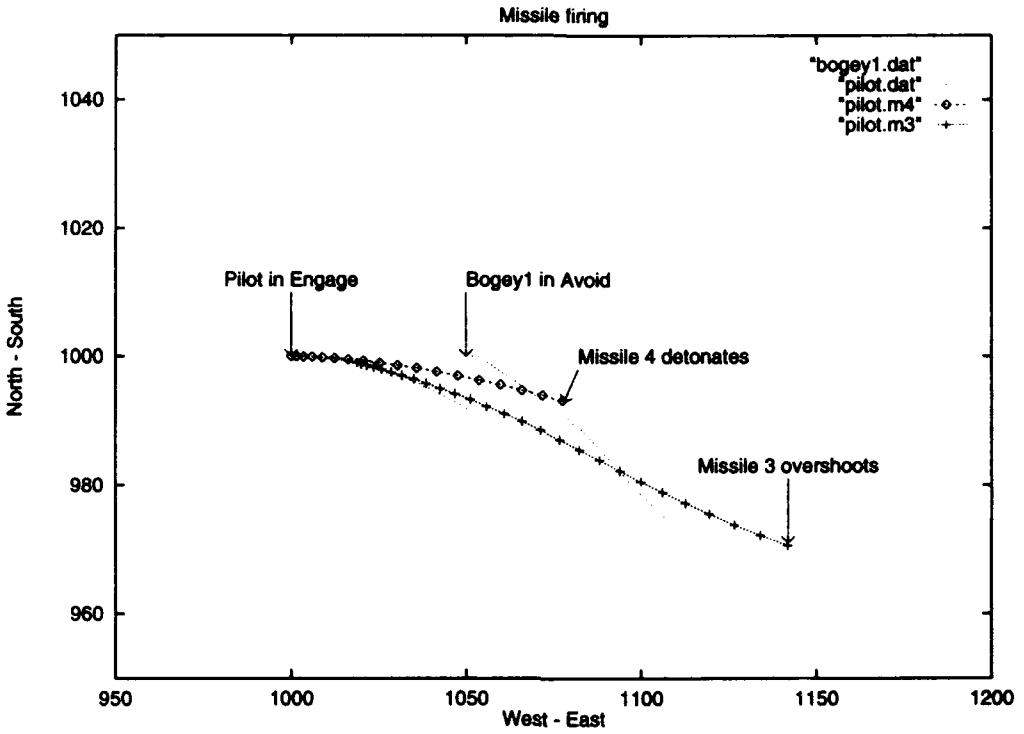


Figure 5.2 **Multiple missile launches.**

The missile load of aircraft in this program is four. Notice that the missile names indicate the number of missiles still existing. After missile 4 detonated, the missile load of *pilot* was decremented by one. After missile 3 detonated, the next missile that *pilot* could use would be missile 2. If *pilot* had a missile load of zero remaining, then he would never be able to switch to Acquire phase.

Although the plot does not show it, the aircraft would both switch to Engage phase after the missile firings and continue their maneuvering.

5.4 *Agent Engagement I*

Figure 5.3 shows a scenario where *pilot* begins in Engage phase behind *bogey1*, who is in Evade phase. *Bogey1* immediately selects an evasive maneuver and attempts to execute a tight turn. *Pilot* continued to pursue but could not decrease his AOT to switch to Acquire phase for a missile shot. *Pilot* overshoots and attempts to turn around for re-engagement. *Bogey1*, though, obtained an advantageous position behind *pilot*. It switched to Acquire phase but could not maintain the advantageous position for the time required for missile preparation. The *bogey1* even attempted a high yo-yo, but was unsuccessful. The remainder of the plot shows the two aircraft circling, each trying to reach an advantageous position.

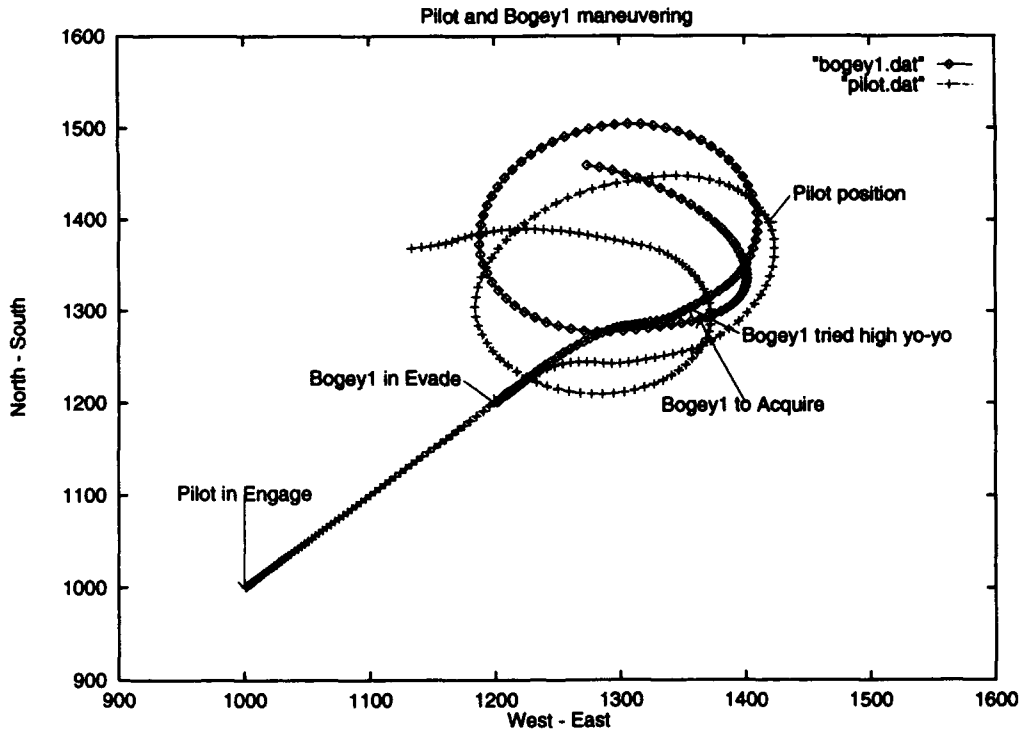


Figure 5.3 **Pilot and bogey1 maneuvering.**

5.5 *Agent Engagement II*

Figure 5.4 shows a scenario where *pilot* begins in Engage phase behind *bogey1*, who is in Evade phase. This scenario is similar to the previous scenario, but the distance between the two aircraft is halved. Once the program started *bogey1* immediately selected an evasive maneuver. It attempted to tighten its turn to decrease *pilot's* AOT. *Pilot* continued to pursue and switched in and out of Acquire phase twice. After *bogey1* turned and lost its disadvantageous position it switched to Engage phase and attempted to engage *pilot.* Figure 5.5 clearly shows the result of each agent attempting to gain an advantageous position. *Pilot* continues to fly in a circle to try to get behind *bogey1* while *bogey1* executes a zoom climb to do the same. During this time they both execute lag, pure, and lead pursuits according to the angles. Eventually *bogey1* reaches a higher altitude and will eventually attempt to turn around.

This plot is also interesting in the fact there is no encoded zoom-climb maneuver rule. *Bogey1*, while attempting to pursue *pilot*, continually climbed at a steep rate to obtain an advantageous position. This continual effort kept *bogey1* at a steep pitch, and resulted in a modified zoom-climb. This zoom-climb, essentially, was the result of a sequence of simple pursuit maneuvers. Ideally, *bogey1* could have reversed at the top of the climb and fire down on *pilot* (31). However, the flight model does not allow such a radical change in pitch.

5.6 *Aberrant Behavior*

Occasionally the agents in the simulation display rather aberrant behavior. Figure 5.6 depicts an example of this anomaly. In this scenario, *pilot* and *bogey1* are both in Engage phase, moving at maximum velocity. The swift approach and violent turning cause both of them to pitch down severely. Since both are flying toward sub-goals established relative to each other, they continue to decrease their altitude. They do not begin to level off until they are well below ground level. This scenario

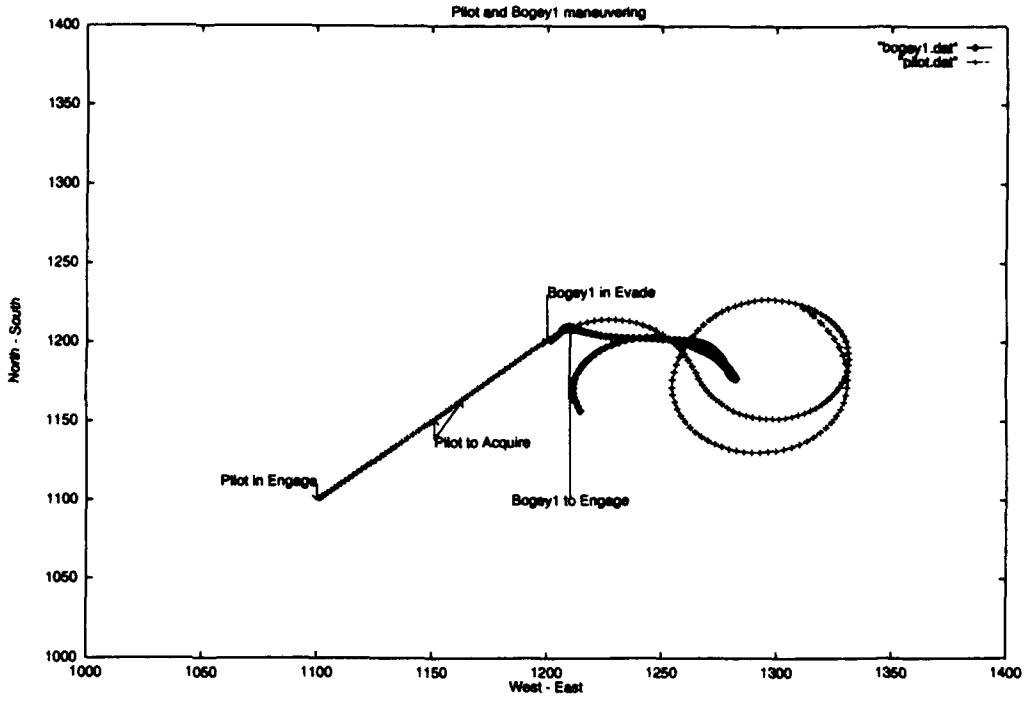


Figure 5.4 **Two agents maneuvering.**

illustrates why over-riding survival rules must be implemented. Maneuvering agents fly relative to each other's position and attitude.

Refinement to the flight model should remedy this situation. Optimal altitude rules do exist, but the agents' severe pitch and maximum velocity reduce the effectiveness of that rule.

5.7 Decision Cycle Execution Time

The initial goal during the development of PDPC was for each decision cycle to equal one second. A simple test was conducted to check how quickly PDPC executes a decision cycle. The test was conducted on a SPARCstation 2, with no other major processes running on the system. A function was added to the plotting function

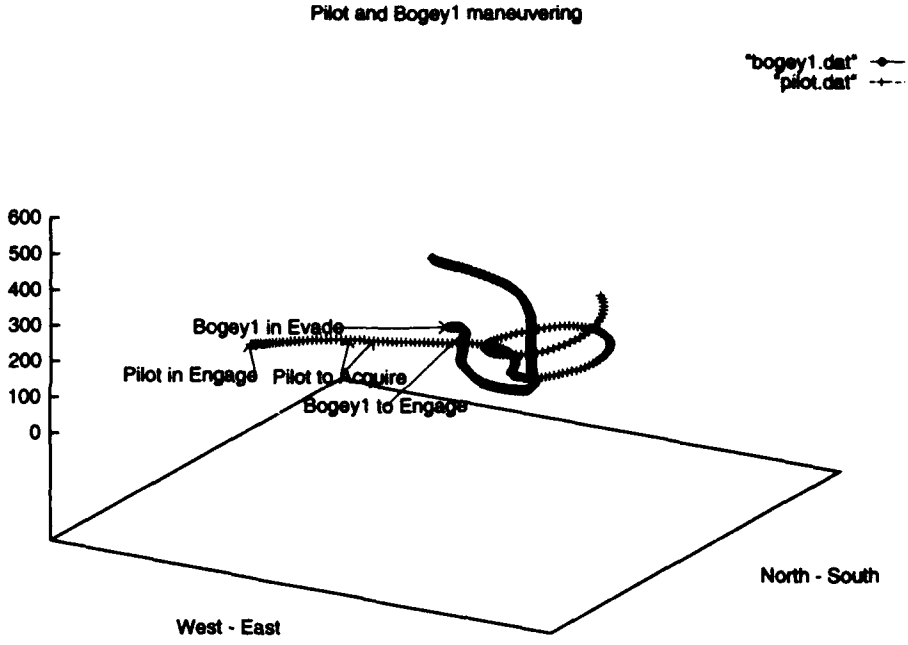


Figure 5.5 **3-dimensional view of two agents maneuvering.**

which added a time stamp each time a new location was written to a data file. The fastest decision cycle only lasted 0.4 seconds; this occurred while the agent was in Pursuit phase. The longest decision cycle lasted 1.9 seconds; this occurred while the agent was in Engage phase. The execution times seem to correlate with the number of rules in a module. Pursuit has only one maneuver, while Engage contains every maneuver. Further statistical analysis is required to establish this correlation.

5.8 Orientation Rates

The behavior of an agent can be radically changed by adjusting its maximum rates of orientation. This enables introduction of agents into the system with specific characteristics. The agents, when maneuvering, always use a combination of roll,

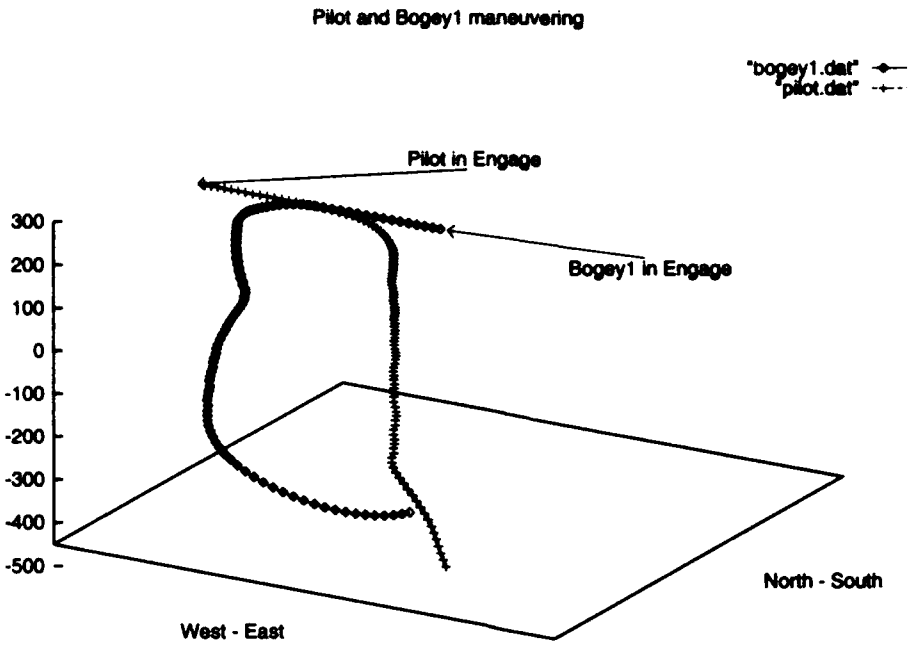


Figure 5.6 **3-dimensional view of aberrant behavior.**

pitch and yaw to fly to a sub-goal. This program does not have the ability to ignore one or more of the orientation components and fly to a sub-goal by adjusting only one of the orientation rates.

Presently the orientation rates are adjusted according to the phase. It would probably be advantageous to adjust these rates according to the phase and maneuver.

5.9 Summary

In this chapter several scenarios of two agents flying were examined. I discussed their phase transitions and maneuver selections. It is clear that the agents are capable of quickly selecting a maneuver and engaging an opponent.

A reactive architecture allows the agents to quickly select a maneuver according to the situation. This simulates a pilot engaging an opponent, since in a dynamic situation such as air combat a pilot must depend on his reflexes for survival. In PDPC agents do not have the flexibility of selecting one of multiple maneuvers. The maneuver knowledge-base is not large enough and there presently is no embedded planner that can choose amongst different maneuvers.

The aberrant behavior can be tamed by encoding more sophisticated rules which can recognize such a situation and by refining the flight model. Presently, orientation rates are adjusted by phase. It would be beneficial to add a mechanism where an agent could turn violently when in it is in a phase that does not normally require it.

VI. Conclusions and Recommendations

6.1 Introduction

In this chapter I present the conclusions of my research and lessons learned. I briefly discuss and summarize the extent of my work. I also list the areas where future work is needed and should continue.

6.2 Research Objective Conclusions

1. Define and create a phase architecture.

   Captain Hipwell and I created a phase architecture that parallels how a pilot acts and maneuvers in air combat. The phase architecture starts from a newly assigned mission and enables an agent to switch between phases to accomplish his goal, and ends with the completion of a mission. It appears that a pilot's mission requirements can be entirely captured by this phase architecture.

   The architecture allows an agent to transition between phases in accordance with the present situation. CLIPS allowed an efficient implementation of the phase architecture by the use of modules, which provide a logical partitioning of rules.

2. Define and create a maneuver architecture.

   It is clear that for a system to approach real-time performance, it must contain a set of reactively defined behaviors. This reactive architecture created used a three level hierarchy for quick response. This is especially important in the domain that we worked in, air combat. Maneuver selection becomes a reflexive behavior. It must be an almost instantaneous transition from sensory input to execution. For behaviors that are not reflexive, or are needed for the future, a higher level planner could be added to supplement the reactive plan. Time constraints did not allow the implementation of such a planner.

It is clear that a rule-based system is one approach out of several that are appropriate for this research effort. CLIPS, and its object oriented extension COOL, was an appropriate system development tool. It enabled us to quickly encode knowledge, and to change to an object oriented approach in mid-stream. It allows programmers who are not domain experts to accurately translate knowledge into rules.

There is an abundance of information in the dynamic world of air combat simulation. Maneuver selection can be successfully accomplished by evaluating several key parameters: target distance, AOT, and target heading and bearing.

3. Use knowledge acquisition techniques to encode a set of maneuvers.

   A complete set of maneuvers have been implemented in this research effort. It is complete in that there is always a maneuver that an agent can execute. Knowledge acquisition techniques were used to gather the domain knowledge. Source selection and identification revealed two convenient, primary sources. These sources were adequate for initial development of the simulation. Further development in this simulation should utilize established human domain experts.

   The maneuvers were encoded as rules which asserted facts for control of calculations and execution. Methodical testing verified and validated the maneuver knowledge-base.

4. Enable an agent to use an opponent's history to predict where he will fly.

   HISTORY objects were instantiated for every agent in the simulation. These tracked the last five locations of each flying agent. Using a least means squares equation, pursuing agents in PDPC could lead a target accurately.

5. Enable an agent to use AI techniques to plan while it has time.

   Hooks for added to enable an embedded planner to influence or suppress the maneuver selection. An embedded planner could also conduct higher level

planning in a variety of areas. The titan ruleset already can make some of these higher level decisions (13).

6.3 Lessons Learned

During the process of creating the knowledge base several lessons were learned. The most important involves the construction of the LHS of rules. The pattern-matching and conditional elements must be constrained so that there is no overlap within a range of values. Rules with overlapping LHS and equal salience within a module can cause nondeterministic behavior, which increases the difficulty of experimentation and testing. Likewise, if the rules test for a range of values, and there are gaps between those ranges, then inadvertent and abrupt interruptions during program execution can occur.

Anyone attempting to develop a simulation that involves flight computations should thoroughly familiarize themselves with the mathematics involved. Flight models depend on an extensive number of calculations and matrix rotations. It is important not to underestimate the impact and time required to create a flight model simulator.

6.4 Future Work

The flight model is not selective in the way it flies. It uses roll, pitch, and yaw to fly to a point. It does not have the capability to use only roll, or only pitch, or only yaw to fly to a sub-goal. It would be beneficial to change the flight model to fly more intelligently.

The addition of a higher-level planner would significantly increase PDPC's abilities. Such a system essentially become a virtual pilot, with both reflexive and reflective abilities present. This would increase the survivability of an agent, since it could plan to prevent extremely hazardous situations from unfolding, especially when planning to attack a target (30).

Another area that requires future work is continued knowledge acquisition to refine the knowledge base. With more time and resources future efforts in this area can add more maneuvers to the knowledge base and verify their correct application. It would be beneficial to consult human domain experts such as pilots for additional maneuver knowledge.

6.5 Summary

This thesis effort established the groundwork for a rule-based, reactive system where agents maneuver against each other in air-to-air combat. It has been shown that autonomous agents can maneuver in an air-combat simulation by utilizing subgoals to induce a particular behavior. This is accomplished by implementing agents as objects, encoding maneuvers as rules, and using facts to control execution. Implementing agents as objects allows the use of object slots to store state information. Encoding the maneuvers as rules creates a fast and efficient reactive structure. Using facts for control allows global control during the simulation. This work shows excellent potential for expansion into a simulation where the agents' behavior can approach that of actual pilots.

Appendix A. Maneuver Computations

A.1 Introduction

This appendix explains PDPC's navigation conventions, important maneuver functions, and orientation calculations. For a detailed explanation of PDPC's flight computations, see Hipwell (13).

A.2 Flight Model Calculations

The flight model manipulates two coordinate systems. The earth ground coordinate (EGC) system is a coordinate system relative to the ground, where the origin is some fixed point. The ABC system is a coordinate system relative to an agent, with the origin being the same as the agent's location. The *move-player* rule performs a series of calculations listed here:

1. Calculates the current heading, climb angle, and roll orientation in EGC space.
2. Rotates the velocity vector onto the nose axis of the agent.
3. Changes the agent's ABC state.
4. Calculates new attitude rates.
5. Calculates new attitude angles.
6. Calculates combined thrust of external and internal forces.
7. Calculates a new acceleration.
8. Calculates a new ABC velocity.
9. Rotates the agent using new attitude angles and produces a new ABC-velocity vector.
10. Reorients the new ABC velocity vector into EGC space.
11. Calculates new orientation angles
12. Decrements fuel.

A.3 Heading Convention

To make our computations easier, we adopted the heading convention shown in Figure A.1. To move north is to move up the y axis and to move east is to move along the x axis. The z axis is used for altitude. Moving up the z axis is to increase altitude. Ground is at z = 0. This is different than normal navigational heading, where north is at 0 degrees. Heading increases in a clockwise direction, which follows normal navigational convention.

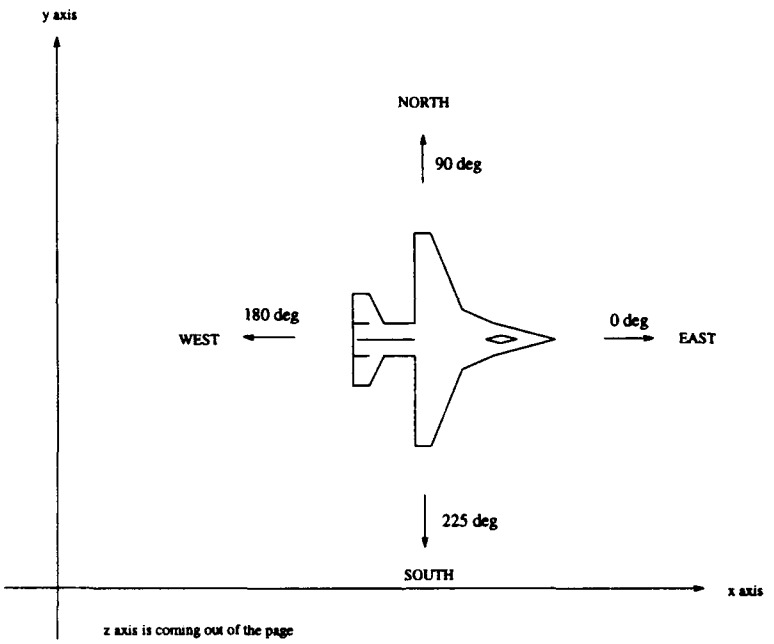


Figure A.1 **Heading convention.**

A.4 Maneuver Calculations

An important calculation that is made in almost every maneuver rule is calculating AOT, which is found using the function *angle-off-the-tail.* This function's arguments are the velocity vector of the target and the location vectors of both air-

craft. Refer to Figure A.2. The aircraft in the middle is the target. Aircraft A's AOT is 0°, which is the minimum value of AOT, while aircraft B's AOT is 180°, which is the maximum value.

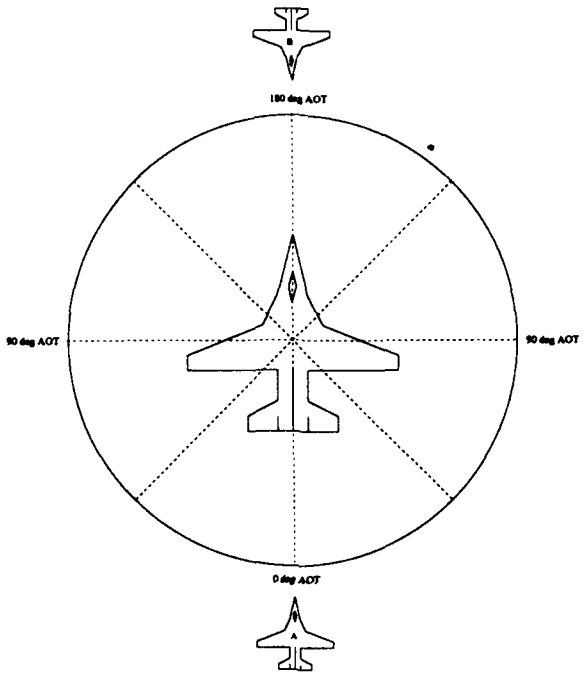


Figure A.2 **Angle off the tail.**

A particularly useful function is *pursuit-possible*. Its arguments are the velocity and location vectors of two aircraft. It returns a positive number of pursuit is possible, and a negative number if pursuit is not possible. Figure A.3 depicts the possible situations two aircraft can be in. In situation 1, aircraft A can pursue aircraft B, but aircraft B cannot pursue aircraft A. In situation 2, both aircraft can pursue each other. In situation 3, neither aircraft A nor aircraft B can pursue.

When calculating a sub-goal for lead pursuit the function *compute-least-squares-point* is used. It uses the last 3 locations of an agent to fit a curve for more accurate

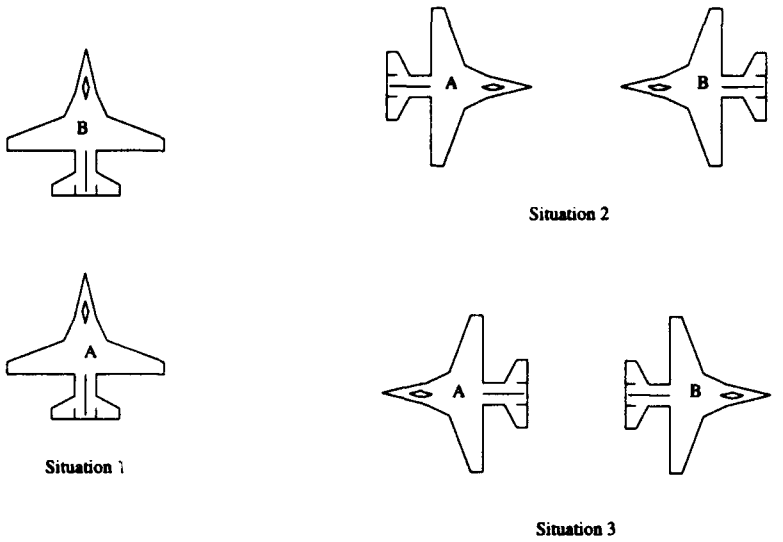


Figure A.3 **Different pursuit positioning.**

lead-point determination. It is possible to use more locations, but that would increase the complexity of the calculations, which would take more time.

A.5 Orientation Calculation

Figure A.4 shows two points in a three dimensional frame. Suppose aircraft A is pursuing aircraft B. The aircrafts' initial orientations using PDPC conventions is zero roll, zero pitch, and a yaw of π. There are three angles that are calculated that are used to determine changes to the roll, pitch, and yaw. The angles calculated are, in Figure A.4, a, b, and c. Angle a is the change in pitch, angle b is the change in yaw, and angle c is the change in roll. If a particular angle is smaller than the allowable orientation rate of the aircraft, then the orientation is changed to that angle. If a particular angle is larger than the allowable orientation rate of the aircraft, then the orientation is adjusted by the allowable rate.

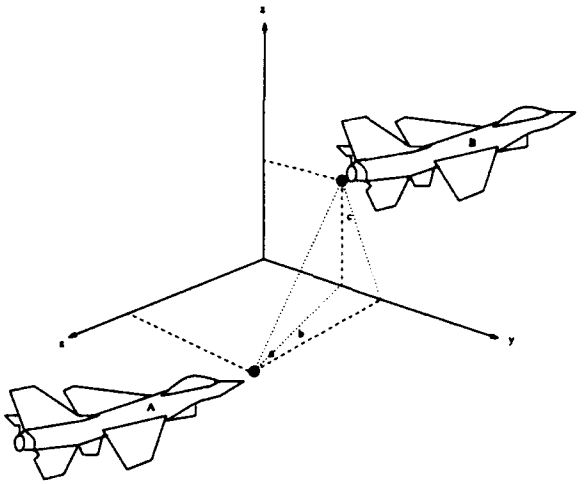


Figure A.4 **Look angles in 3-dimensional space.**

The standard orientation rate in this simulation is used while an agent is in Cruise phase. The orientation rates in the maneuver phases are calculated by multiplying the Cruise rates with a constant. This constant was derived through experimentation. Changing the Cruise rates will affect an agent's behavior in all the other phases. The file basicset.col contains the orientation values and the constants used to derive orientation rates for all the phases.

Appendix B. Rulebase and Functions Listing

This appendix lists the rules and functions by module (phase). The phases are listed in alphabetical order, except for the MAIN constructs, which are listed last.

A complete source code listing may be obtained from:

MAJ Gregg Gunsch
AFIT/ENG (AI Lab)
2950 P St
Wright-Patterson AFB, OH 45433-7765

B.1 Acquire

The following rules are in acquire.col.

Phase changes While in Acquire phase the agent can switch to Fire, Breakoff or Engage.

- leave-acquire-to-fire
- leave-acquire-to-breakoff
- leave-acquire-to-engage

Maneuver decision All pursuits are available in Acquire phase. Note there are no forward quarter maneuvers.

- decide-lag-enemy
- decide-lag-enemy-parallel
- decide-pure-enemy
- decide-lead-enemy

Sub-goal calculations The appropriate sub-goal rules.

- decide-on-lag-point
- decide-on-pure-point

- decide-on-lead-point

Speed control The agent can decrease, match, or increase speed.

- speed-slower
- speed-same
- speed-faster

Planner hook This rule will check a maneuver plan and adjust the sub-goal if necessary.

- check-maneuver-plan

B.2 Analyze

The following rules are in analyze.col.

Phase changes From Analyze phase the agent can switch to Breakoff, Search, or Engage.

- leave-analyze-to-breakoff
- leave-analyze-to-search
- leave-analyze-to-engage-I
- leave-analyze-to-engage-II

Maneuver decision All pursuits are available. No forward quarter maneuvers available.

- decide-lag-enemy
- decide-lag-enemy-parallel
- decide-pure-enemy
- decide-lead-enemy

Sub-goal calculations Appropriate sub-goal rules.

- decide-on-lag-point
- decide-on-pure-point
- decide-on-lead-point

Speed control The agent can decrease, maintain, or increase speed.

- speed-slower
- speed-same
- speed-faster

B.3 Avoid

The following are in avoid.col.

Phase changes From Avoid phase the agent can switch to Search or Engage.

- leave-avoid-to-search
- leave-avoid-to-engage-I
- leave-avoid-to-engage-II

Maneuver decision Only one maneuver necessary in Avoid phase.

- defend-against-missile-pursuit

Sub-goal calculations The appropriate rule for sub-goal calculation.

- decide-on-jink-point

Speed control In Avoid we want the agent to move as quickly as possible.

- speed-faster

Planner hook This rule will check a maneuver plan and adjust the sub-goal if necessary.

- check-maneuver-plan

B.4 Breakoff

The following are in breakoff.col.

Phase changes From Breakoff phase the agent can switch to Engage or Disengage phase.

- leave-breakoff-to-disengage
- leave-breakoff-to-engage

Maneuver decision Only maneuver in breakoff, and that is to leave the general vicinity away from the enemy.

- breakoff-from-engagement

Sub-goal calculation Only one rule necessary.

- decide-on-run-point

Speed control Only one speed: fast.

- turn-and-run-faster

B.5 Cruise

The following rules are in cruise.col.

Phase changes From Cruise phase the agent can switch to Identify, Refuel, Recall, and Landing.

- cruise-to-identify

- cruise-to-refuel
- cruise-to-recall
- cruise-to-landing

Cruise behavior The following rules control how an agent flies in Cruise phase.

- cruise-to-waypoint
- end-cruise-phase

B.6 Disengage

The following are in disengag.col

Phase changes From Disengage phase the agent can switch to Retreat, Refuel, or Recall phase.

- disengage-to-retreat
- disengage-to-refuel
- disengage-to-recall

Individual These rules enable a follower or leader to independently disengage and also check the titan rules.

- disengage-follower
- disengage-leader
- disengage-to-titan
- check-disengage-criteria

B.7 Engage

The following are in engage.col.

Phase changes From Engage phase the agent can switch to Evade, Acquire, or Pursuit phase.

- leave-engage-to-evade-I
- leave-engage-to-evade-II
- leave-engage-to-acquire
- leave-engage-to-pursuit

Maneuver decision The maneuver decision is made according to the present situation. Each decision cycle the situation is quickly reassessed, and a change is made if necessary.

- decide-lag-enemy
- decide-lag-enemy-parallel
- decide-pure-enemy
- decide-lead-enemy
- decide-lead-enemy-side
- nose-to-nose-turn
- decide-on-lead-turn
- pursuit-with-separation
- decide-on-lag-roll
- decide-high-yo-yo
- decide-on-default-maneuver

Sub-goal calculations Engage phase has many sub-goal rules to match the maneuvers.

- decide-on-lag-point
- decide-on-pure-point

- decide-on-lead-point
- decide-on-nose-turn-point
- decide-on-lead-turn-point
- decide-on-sep-point
- decide-on-lag-roll-point
- decide-on-high-yo-point

Speed control The aircraft's speed is determined by the sub-goal distance. If it is too close, the aircraft will slow down. If too far, the aircraft will speed up.

- speed-slower
- speed-same
- speed-faster
- turn-speed-same

Additional rules There are some additional rules implemented.

- check-for-optimal-altitude

Planner hook This rule will check a maneuver plan and adjust the sub-goal if necessary.

- check-maneuver-plan

B.8 Evade

The following are in evade.col.

Phase changes From Evade phase the agent can switch to Engage only.

- leave-evade-to-engage-I
- leave-evade-to-engage-II

Maneuver decision All the defensive maneuvers are implemented in Evade phase.

- defend-against-lag-pursuit-I
- defend-against-lag-pursuit-II
- defend-against-pure-pursuit-I
- defend-against-pure-pursuit-II
- defend-against-pure-pursuit-IIa
- defend-against-lead-pursuit-I
- defend-against-lead-pursuit-II
- defend-against-lead-pursuit-IIa

Subgoal calculations Appropriate sub-goal rules.

- decide-on-turn-point-I
- decide-on-turn-point-II
- decide-on-turn-point-IIa
- decide-on-turn-point-III

Speed control The agent can decrease or increase speed.

- turn-and-speed-faster
- turn-and-speed-slower

B.9 Fire

The following are in fire.col.

Phase changes From Fire phase the agent can switch to Acquire, Breakoff, Engage, or Analyze phase.

- leave-fire-to-acquire

- leave-fire-to-breakoff
- leave-fire-to-engage
- leave-fire-to-analyze

Maneuver decision No forward quarter maneuvers here.

- decide-lag-enemy
- decide-lag-enemy-parallel
- decide-lead-enemy

Sub-goal calculations Appropriate sub-goal rules.

- decide-on-lag-point
- decide-on-pure-point
- decide-on-lead-point

Speed control The agent can decrease, maintain, or increase speed.

- speed-slower
- speed-same
- speed-faster
- turn-speed-same

Missile rules Here are the various rules for a missile. The rules check for detonation and fuel use. The missile uses only one maneuver, pure pursuit.

- check-for-missile-destruct
- check-missile-life
- decide-missile-pure-pursuit
- missile-pure-speed

Planner hook This rule will check a maneuver plan and adjust the sub-goal if necessary.

- check-maneuver-plan

B.10 Pursuit

The following are in pursuit.col.

Phase changes From Pursuit phase the agent can switch to Evade or Engage phase.

- leave-pursuit-to-evade-I
- leave-pursuit-to-evade-II
- leave-pursuit-to-engage

Maneuver decision Only one maneuver used in Pursuit phase.

- long-distance-lead-pursuit

Sub-goal calculation Only one sub-goal rule.

- decide-on-lead-point

Speed control Normally, the pursuing aircraft will fly at maximum speed to close the distance to its target as quickly as possible.

- pursuit-lead-speed

B.11 Search

The following rules are in search.col.

Phase changes From Cruise phase the agent can switch to Identify, Refuel, and Recall.

- search-to-identify
- search-to-refuel
- search-iterations

Titan checks The following rules check the titan rules for information.

- search-to-titan
- check-radar
- check-radar-mode

Search behavior The following rules control an agent's behavior in Search phase.

- follower-search
- CAP-station-to-patrol

B.12 MAIN

Airmath functions The following functions are used to calculate flight specific parameters.

The following functions are in airmath.col.

- Vector math functions
  - bearing-arctangent
  - calculate-angles
  - altitude
  - speed
  - climb-rate
  - counter-force
- Navigation functions
  - heading

- climb-angle
- roll-angle
- target-range
- target-distance
- line-of-sight
- target-bearing
- angle-off-the-nose
- angle-off-the-tail
- target-aspect-angle
- track-crossing-angle
- separation
- colinear-p
- closing-velocity

• Flight dynamics vector functions

- EGC-to-ABC
- ABC-to-ABC
- ABC-to-EGC
- change-attitude-moments
- change-attitude-rates
- change-abc-attitude
- change-thrust
- change-abc-acceleration
- change-orientation
- change-velocity
- change-location

Maneuver functions The following functions are used by the maneuver rules to determine certain conditions.

The following are in basicmat.col.

- convert-positive-angle
- limit-roll
- create-lag-point
- create-lead-point
- create-jink-point
- create-lead-turn-point
- create-sep-point
- create-lag-roll-point
- create-yo-point
- create-trail-point
- create-echelon-point
- create-abreast-point
- find-least-squares-equation
- find-least-squares-point
- compute-least-squares-point
- choose-a-turn
- speed-slower
- speed-same
- speed-faster
- speed-control
- speed-match

- find-los
- pursuit-possible

Maneuver flight functions The following functions are used by the orientation rule to decide on a new orientation.

The following functions are in basicset.col.

- limit-to-90
- calculate-yaw
- calculate-pitch
- calculate-roll
- decide-on-yaw
- decide-on-pitch
- decide-on-roll
- decide-on-roll-limit
- decide-on-pitch-limit

Bibliography

1. Anderson, Tracy L. and Max Donath. "A Computational Structure for Enforcing Reactive Behavior in a Mobile Robot." *Mobile Robots III; Proceedings of the Meeting, Society of Photo-Optical Instrumentation Engineers 1007*. 370–382. 1988.
2. Brooks, Rodney A. "A Robust Layered Control System For A Mobile Robot," *IEEE Journal of Robotics and Automation*, 979–984 (March 1986).
3. Chrystall, K., et al. "A Robotic Planning System." *Proceedings of the Sixth CASI Conference on Astronautics*. 305–313. 1990.
4. Culp, Donald R. "Interpretation of Space Shuttle Telemetry." *First CLIPS Conference Proceedings I*. 290–304. 1990.
5. Dean, Thomas and Mark Boddy. "An Analysis of Time-Dependent Planning." *Proceedings of the National Conference on Artificial Intelligence*. 49–54. AAAI, 1988.
6. Department, NASA Software Technology. *CLIPS 6.0 Basic Programming Guide*. National Aeronautical Space Administration, 1993.
7. Dodhiawala, Rajendra, et al. "Real-Time AI Systems: A Definition and an Architecture." *Proceedings of the International Joint Conference on Artificial Intelligence*. 256–260. Los Altos, California: Morgan Kaufmann, 1989.
8. Dyer, Douglas E. and Gregg H. Gunsch. "Enlarging the Universal Plan for Air Combat Adversaries." *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*. 255–261. 1993.
9. Firby, R. James. "Building Symbolic Primitives with Continuous Control Routines." *Proceedings of the 1st International Conference on Aritificial Intelligence Planning Systems*. 62–69. 1992.
10. Georgeff, M.P., et al. "Reasoning and Planning in Dynamic Domains: An Experiment with a Mobile Robot." *Proceedings of the Workshop on Space Telerobotics 3*. 27–39. July 1987.
11. Gervasio, Melinda T. "Learning General Completable Reactive Plans." *Proceedings of the National Conference on Artificial Intelligence*. 1016–1021. 1990.
12. Haley, Paul. "Data-Driven Backward Chaining." *Second CLIPS Conference Proceedings*. 325–331. 1991.
13. Hipwell, Dean. *Developing Realistic Cooperative Behaviors for Autonomous Agents in Air Combat Simulation*. MS thesis, Air Force Institute of Technology, 1993. AFIT/GCE/ENG/93D-05.
14. Hluck, George S., "Notes from the Third Conference on Computer Generated Forces and Behavioral Representation," March 1993.

15. Homeier, Peter V. and Thach C. Le. "ECLIPS: An Extended CLIPS for Backward Chaining and Goal-Directed Reasoning." *Second CLIPS Conference Proceedings*. 273–283. 1991.

16. Jones, Randolph M., et al. "Intelligent Automated Agents for Flight Training Simulators." *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*. 33–42. 1993.

17. Kaebling, Leslie Pack. "An Architecture for Intelligent Reactive Systems." *Reasoning About Actions and Plans* edited by M. Georgeff and A.L. Lansky, 395–410, Morgan-Kaufmann, 1987.

18. Kreutzer, Wolfgang and Bruce McKenzie. *Programming for Artificial Intelligence*. Addison-Wesley Publishers Ltd., 1991.

19. Le, Thach and Peter Homeier. "Portable Inference Engine: An Extended CLIPS for Real-Time Production Systems." *2nd Annual Workshop on Space Operations Automation and Robotics*. 187–192. 1988.

20. Manouchehri, Davoud, et al. "Autonomous Robotic Systems for SEI Tasks." *Proceedings of the 28th Space Congress*. 19–26. 1991.

21. Mezera, David. *Using Discovery-Based Learning to Improve the Behavior of an Autonomous Agent*. MS thesis, Air Force Institute of Technology, 1993. AFIT/GCE/ENG/93D-10.

22. Mitchell, Tom M. "Becoming Increasingly Reactive." *Proceedings of the National Conference on Artificial Intelligence*. 1051–1058. AAAI, 1990.

23. Nazareth, Derek L. "Issues in the verification of knowledge in rule-based systems," *International Journal on Man-Machine Studies*, 255–271 (March 1989).

24. Payton, David W., et al. "Plan Guided Reaction," *IEEE Transactions on Systems, Man, and Cybernetics*, 1370–1382 (November/December 1990).

25. Rich, Elaine and Kevin Knight. *Artificial Intelligence* (2nd Edition). McGraw-Hill, Inc, 1991.

26. Rosenschein, Stanley J. and Leslie Pack Kaebling. "Integrating Planning and Reactive Control." *Proceedings of the NASA Conference on Space Telerobotics2*. 359–366. Palo Alto, California: Teleos Research, June 1989.

27. Russell, Stuart J. and Shlomo Zilberstein. "Composing Real-Time Systems." *Proceedings of the International Joint Conference on Artificial Intelligence*. 212–217. Los Altos, California: Morgan Kaufmann, 1991.

28. Schoppers, Marcel J. "Universal Plans for Reactive Robots in Unpredictable Environments." *Proceedings of the International Joint Conference on Artificial Intelligence*. 1039–1046. Los Altos, California: Morgan Kaufmann, 1987.

29. Schoppers, Marcel J. "Building Plans to Monitor and Exploit Open-Loop and Closed-Loop Dynamics." *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems*. 204–213. Los Altos, California: Morgan-Kaufman, 1992.

30. Secarea, Jr., V.V. and H.F. Krikorian. "Adaptive Multiple Target Attack Planning in Dynamically Changing Hostile Environments." *Proceedings of the IEEE National Aerospace and Electronics Conference*. 1117–1123. 1990.

31. Shaw, Robert L. *Fighter Combat*. Naval Institute Press, 1985.

32. Systems, Titan. *Pilot's Decision Definition and Analysis*. Technical Report AFWAL-TR-86-0002, Air Force Wright Aeronautical Laboratories, 1986.

33. Whitehead, Steven D. and Dana H. Ballard. "Reactive Behavior, Learning, and Anticipation." *Proceedings of the NASA Conference on Space Telerobotics5*. 333–344. January 1989.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1993	Master's Thesis

4. TITLE AND SUBTITLE
DEVELOPING REALISTIC BEHAVIORS IN ADVERSARIAL AGENTS FOR AIR COMBAT SIMULATION

5. FUNDING NUMBERS

6. AUTHOR(S)
George S. Hluck, Captain, USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/93D-06

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis describes an initial effort into creating a rule-based, reactive system for air combat simulation. This program uses the object-oriented extension of the expert system tool known as the C Language Integrated Production System (CLIPS). This effort rose out of the need for creating and integrating semi-autonomous forces for the Distributed Interactive System (DIS).

This thesis describes the basic maneuvers a pilot uses in present air-to-air combat. The methodology includes the design decisions, knowledge-base development, phase architecture, and maneuver architecture development. The actual implementation of the selected architecture is described. This thesis also discusses the results of experimental runs with two agents maneuvering against one another.

14. SUBJECT TERMS	15. NUMBER OF PAGES
ARTIFICIAL INTELLIGENCE; AIR COMBAT; SIMULATION; KNOWLEDGE-BASED; CLIPS	108
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL