IMPLEMENTING A DOGFIGHT ARTIFICIAL PILOT

Mahery ANDRIAMBOLOLONA
Pascal LEFEUVRE

Research Report DKS03-07 / ICE 07
Version 0.9, July 2003

Mediamatics / Data and Knowledge Engineering group
Department of Information Technology and Systems
Delft University of Technology, The Netherlands

TU Delft

Abstract

Since aircraft capabilities and information complexity have increased, flying an airplane has become more complex. Consequently, pilots run the risk of information overload and making mistakes. That is why situation awareness is usually considered to be one of the most important aspects when flying an airplane. Thus, a number of systems have been developed to help prevent mistakes while flying by increasing the situation awareness of the pilot. These systems filter the huge amount of information for the pilot and provide him with only relevant information based on the current situation the pilot is in and take into account the workload of the pilot.
This report tries to point out how human behavior can be modeled in order to realize an automated pilot (called bot) that acts like a human one. Many automated pilots already exist. However the purpose here is the human resemblance. The system designed for the project focuses on the decision-making process of a bot during a dogfight.
First the domain knowledge is described that has been gathered about how a real pilot behaves while dog fighting. That is to say which data it takes into account and how it does it in order to choose and execute the right maneuver. A cognitive model is then presented that defines how information is processed and how decision is taken. A first prototype, which implements this cognitive model, has been realized. It stands as the first step of an iterative approach to improve the behavior of the bot. Decision tree processes are run to take decision. Since the results of the prototype were not good, a second release has been carried out. Many improvements have been added and the decision trees processes have been kept. Running the decision tree processes leads to several possible maneuvers. In this release a way to dismiss incompatible maneuvers has been added in order to elect the most suitable one. The results for this second release are not yet available.

Acknowledgments

This 5-month placement in TU Delft had brought us about working with people we would like to thank. First and foremost are Dr. drs. Leon Rothkrantz and ir. Patrick Ehlert who supervised and advised us during the entire project. Moreover, all our greetings for Quint Mouthaan and Marten Tamerious, two colleagues whose knowledge, ideas and advice helped us a lot to carry out this project.

TABLE OF CONTENTS

Chapter 1 Introduction

The ones that can have a look to airplanes throughout the last century will be impressed by the way it has become increasingly complicated to fly. When flying, a pilot has to manage a huge amount of information provided by complex instruments, gauges and his eyes. Consequently, pilot workload has increased dramatically and can lead to overload.
To face this problem several research projects have been carried out. All of them try to find a solution to reduce the pilot workload. The Intelligent Cockpit Environment (ICE) project stands as one of them. The purpose is to find ways to automate pilots' tasks. One of the ICE project field is to develop a fully autonomous (computer-driven) pilot. This flight bot will be able to perform almost all the tasks a real pilot can do. A real pilot that uses this bot can draw conclusions about the way to make decisions. It can be also useful for finding out the situation with the most workload. Thus research for automating tasks can be carried out on these situations.

1.1 The Ice project

Situation awareness is usually considered to be one of the most important aspects when flying an airplane. Research shows that many human errors in aviation are caused by a lack of situation awareness [Ehlert03]. Since aircraft capabilities and information complexity have increased, pilots run the risk of information overload. Thus, a number of systems have been developed to help prevent mistakes while flying by increasing the situation awareness of the pilot. These systems filter the huge amount of information for the pilot and provide him with only relevant information based on the current situation the pilot is in and take into account the workload of the pilot.
The **I**ntelligent **C**ockpit **E**nvironment (**ICE**) project deals with new techniques and technology for automation and intelligent human-machine communication in the cockpit. The idea is that a Crew Assistance System (CAS) helps the crew with their tasks by offering relevant information, taking over tasks, or prioritising alerts in case of malfunctions or mistakes. Thus the crew situation awareness and performance will be increased.
To help the crew, the CAS should form its own ideas about what is going on, similar to the crew's situation awareness. Also, it should be capable of reasoning and making decisions what to do.
Until now, some projects have already been carried out in the ICE project. For instance a neural-network-based system has been developed to try to compute what the pilot is doing. Instead of encouraging results what has been pointed out is the lack of training and test data so that it is impossible to train the neural network completely. Moreover a system has been developed that can detect which situation is currently occurring during a flight. This is a probabilistic model. It uses a knowledge base containing data that describes a number of situations that the pilot might encounter. Some research has also been carried in the field of autonomous flight bots. These flight bots are designed to imitate the behaviour of a human. For instance one project deals with a system that generates and executes a flight script based on a flight plan for a Cessna (a kind of plane).
Eventually, the ICE project goal is to design, test, and evaluate computational techniques that can be used in intelligent situation-aware crew assistance systems development. Using methods from artificial intelligence, ICE focuses primarily on data fusion, data processing and the reasoning part of these systems. Special issues addressed in the ICE project are:

- Situation recognition
- Mission or flight plan monitoring
- Attack management
- Pilot workload monitoring

1.2 Related projects

The first efforts to design a crew assistance system date back to the mid 1980s. Although today, several prototypes CASs exist and a lot of (theoretical) research has been done in this area, in practice there has been little success. In the following section, we will give a short description of some related research projects. For more details, refer to [Ehlert03].

Pilot's Associate

The Pilot's Associate is a program initiated by the United States Air Force's Wright Laboratory and the Defense Advanced Research Projects Agency (DARPA) in 1986. The goal of the PA project was to enhance a military fighter pilot's situation awareness, enabling him to make better decisions. The Pilot's Associate (PA) architecture basically consists of a set of cooperating knowledge-based subsystems that can communicate with each other. It can manage uncertainty, adapt the information provided to the pilot depending the current situation and recognize the pilot intention and detect pilot errors.
Some problems occurred. Because of the developers-team multiplicity, the gathering of the different parts of the project was laborious. Finally, after two demonstrations, the system showed it had difficulties performing in real time. Problems arose also concerning knowledge acquisition and validation.

Rotorcraft Pilot's Associate

The Rotorcraft Pilot's Associate (RPA) is a five-year program of the US Army Aviation Applied Technology Directorate. It extends the Pilot's Associate project by applying it to an attack/scout helicopter instead of a jet aircraft.
Simulation studies with the RPA CIM (Cockpit Information Manager) have showed that even though the system is not always correct in its assessments, when giving a choice the test pilots prefer flying with all the CIM's features turned on.

CASSY

CASSY is a crew assistance system developed by the Flight Mechanics and Flight Guidance group of the University of the German Armed Forces in Munich Germany in cooperation with the Dornier Company. The purpose of CASSY is to guide the pilot with the objectively most urgent task or subtask, while avoiding overload of the crew in planning, decision-making and execution. CASSY primarily focuses on flight planning and pilot error detection in civil (transport) aircraft under instrumental flight rules (IFR). The system permanently shows its assessed situation to the crew and only becomes active when problems are detected.
A working prototype of CASSY has been tested in-flight in a civil transport airplane. The responses of the test pilots were very positive, even though only a part of the system was functional.

CAMA

CAMA, which stands for Crew Assistant Military Aircraft, extends the CASSY system to military transport aircraft. The German Ministry of Defense initiated the CAMA project in 1992 and the system was based on the experiences with CASSY.
The architecture of CAMA is based upon CORBA and allows applications to communicate with each other no matter where they are located or who has designed them. CORBA servers provide the knowledge that is being processed by CORBA clients that form the "cognitive" tasks. CAMA uses a state-based approach (Petri-nets) to model the normative pilot actions that follow from the regulations in the aviation handbooks. Case-based reasoning is used to evaluate the pilot's actual actions and adapt state-transitions so individual pilot's differences can be learned.
In 1997 and 1998 the first flight simulator test runs with CAMA were performed. In 2000 four separate pilots performed five real life test flights. Results appeared to be successful.

1.3 Project definition

1.3.1 Project goals

The goal of the project is the design and implementation of an artificial player, also called bot, for a flight simulator that should behave like a human pilot. Firstly it should be able to fly without human supervision. It means that once a destination is given, the bot pilot should reach its goal without any human intervention. Secondly, its behavior can be adapted to unexpected events such as another plane approaching or a missile launched. The final goal is to achieve a complete artificial pilot that really looks like a human player so that it is impossible to make the difference.

1.3.2 Project borders / Requirements

Designing the entire bot is not trivial. For instance, one student worked on the development of an advanced Quake bot a few years ago [QuakeBot01]. It took 4 years. So it is important to assess what is feasible within 5 months and what is not.
The initial project description has been clarified and bordered in order to propose a relevant and feasible project. The resulting project is "Flight simulator: implementing a **dogfight artificial pilot**".
This section deals with all the Artificial Player requirements for the project. Because the current project is part of the "Flight Simulator: implementing **an artificial pilot**" project, some global requirements are first specified. Then additional requirements, specific to the dogfight model, are written down.
The functionalities an artificial pilot should implement are first shown. Then the functionalities that will be considered for the project are specified those. The choice has been lead considering the time constraint (5 month) and the lack of information (expert knowledge are required).

The functionalities an artificial pilot should implement has been pointed out while considering how a real pilot reasons. It consists in a five-step process:

1. The pilot observes and gets information;
2. The pilot interprets the information to assess what is happening;
3. Considering what he assessed, he decides what he has to do, he defines his goals;
4. To reach his goals, the pilot now defines which actions he has to perform;
5. He executes the actions and gets new observation and information;
6. Finally the bot is able to evaluate the results of his action behavior.

Considering this cognitive model, the bot's functionalities have been identified. All these functionalities are illustrated in the following diagram.

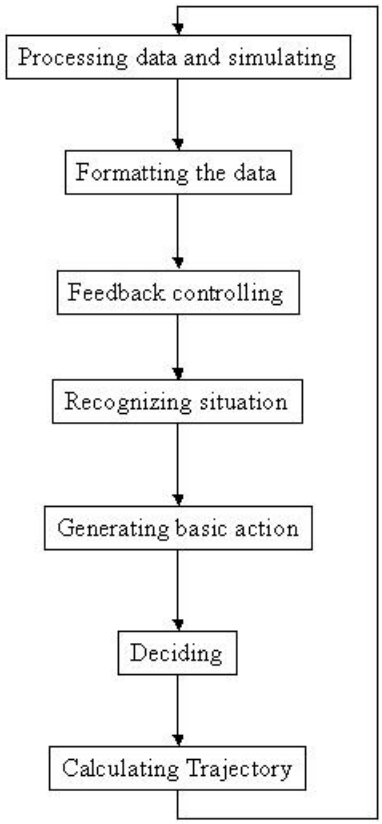


Figure 1: functional diagram

First the Flight simulator engine provides some data. The formatting data block translates these rough data so that they are understandable by the pilot model. A pilot model for a flight simulator is a program that can fly an aircraft as a human does. The situation recognizer can process them. Information about the current situation is communicated to the decision making process. A decision is made and the resulting trajectory is generated. A sequence of geographical locations is computed. This information is processed and translated so that the flight simulator engine can understand them.

Follows a description of these functionalities:

- **Processing data and simulating:** as inputs, it deals with taking data provided by any kind of players, included a bot, generates outputs data and simulates the scenario. It is not a functionality of the bot itself. It directly deals with the flight simulator engine itself;
- **Formatting the data:** it deals translating and formatting rough data from the Flight simulator engine so that the pilot model can understand them.
- **Feedback controlling:** this functionality consists of comparing the data from the flight simulator engine to the data expected. For instance it compares the current aircraft position to the location expected. If too far, some corrections are suggested.

- **Recognizing the situation**: Among a huge amount of data, the artificial player should recognize some standard situations a real pilot usually addresses. By data it is meant data coming the Flight simulator engine that have been formatted;
- **Deciding:** it is the decision-making process. It consists in processing several kinds of data and adopting the most suitable behavior. By data it is meant both processed data by a situation recognizer and rough data coming from the Flight Simulator engine.
- **Calculating trajectories:** with a specific maneuver as input, it computes the route the aircraft should follow. That is to say a sequence of locations.
- **Generating basic actions:** transforming macro actions or orders in sequence of basic actions that are directly understandable by the flight simulator engine.

The project will focus on **implementing the decision making process** of the bot. A specific part of the decision making process will be developed. Working on the dogfight decision-making process is challenging but remains a hard task. Focusing on the attacker behavior is the main issue of the project. It seems to be a realistic goal to achieve within our project period. Here are the requirements concerning the dogfight bot specific functionalities:

Develop the dogfight decision process	Given a situation the program should be able to choose the most suitable maneuver.

To achieve these goals, here are the main assumptions:

Only modelling the attacker behavior	It is assumed that a defender model has already been developed. It can maneuver independently. Any reversal situation is beyond the scope of this project. The model has to deal with offensive maneuvers only.
Model gun attacks	The only weapons used in the dogfight are guns.
Compute a decision if some information is missing	To simplify in a first approach, it is assumed that all information are provided to the model.
Assess situation	Recognizing the situation from all the rough parameters is not the matter of this project. It will be assumed that this module is operational. How to communicate with it is not yet defined.
Assess the enemy's trajectories	It is supposed to be developed. In fact, developing such functionalities is beyond the scope of this project as too much information is missing. It is assumed that a module has been designed and implemented. During the project it will be assumed that these functionalities are operational.
Calculate trajectories	It uses skills that a pilot has acquired through practice. An artificial player has a memory to reason but data are required. Such rough data as maneuver positions are difficult to find. That is why it is supposed that this functionality is operational.

Here are the requirements about the architecture program:

Interface with the Flight simulator engine	The decision making process is independent from the flight simulator used. That is why such blocks, as *formatting data* and *generating basic actions* will not be developed.
Make it easy to modify	One should be able to adjust parameters or algorithms without the intervention of computer science experts.
Be flexible	It should be documented enough that anyone that did not participate to the project development can modify it in a short time

1.3.3 Related Projects

Improving the behavior of individual automated entities in simulation exercise is a real challenge. Simulation is cheaper and more flexible. Many projects have been developed. Two of them have been mentioned below because they illustrate two different approaches to address the following common issue: *getting as close as possible to the human behavior*. In the first one, researchers and engineers consider that they have enough information to specify and encode the model. Thus, the model is complex. In the second approach, people involved realize how crucial it is for the model to be specified and encoded by air-combat experts. The testing and prototyping phases are quicker and more relevant. The following examples illustrate both approaches.

On one hand, there is the TacAir-Soar project that was deployed at the WISSARD facility at the Oceana Naval Air Station and the Air Force Research Laboratory at Mesa, Arizona. It is an intelligent rule-based system that generates “human like” behavior. It can execute most of the airborne missions. The specificity of this system is that system experts have encoded the model behavior. Other main feature is how complex it is since about 5200 rules are executed.

On the other hand, a project was lead at Linköping University in collaboration with Saab Military Aircraft AB, Sweden [Corad97]. It deals with intelligent agents for the air domain (beyond range combat). The main requirement of this agent is that it should be simple enough so that experts of the air-combat domain could encode model behavior without system expert help. It is also based on production rules.

As far as we are concerned, a lot of information is missing. It would be unwise to claim that we could design a model as close as possible to a real pilot. Consequently the project approach will be closer to the second project.

Chapter 2 Domain Knowledge

Designing an artificial pilot requires information about how a real pilot flies. This chapter aims to present what information has been collected about the real pilot decision making process. It also tries to provide some precise ideas about the way the attacker pilot model can take inputs into account to choose the most accurate maneuver to perform. This can be used when modeling the bot to make it as close as possible to a real pilot.

It starts giving an overview of dogfight context: the first section provides a short description of some common maneuvers used. The second section presents the principles used by a real pilot to choose the most accurate maneuver. It also deals with the steps followed to intercept the opponent.

2.1 Description of maneuvers

2.1.1 Principles of Basic aircraft Maneuvers

This section deals with principles of Basic aircraft maneuvers. The bot should be aware of these principles. They are presented out of any tactical context. Fight maneuvers are designed for tactical purposes. They are the topic of the next section.

- Roll

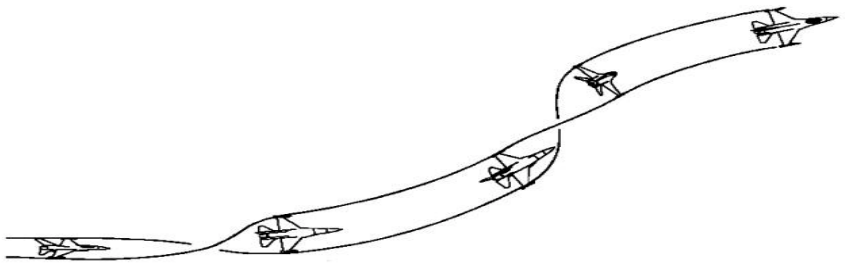


Figure 2: Roll

Roll is a basic aircraft movement. It consists of rotating along the longitudinal axis. That is this line that extends along the fuselage from nose to tail. As Rolling allows the pilot to position his lift vector, slows the forward velocity of the aircraft -The lift vector is a vector that sticks directly out of the top of the jet, perpendicular to the aircraft's wings.

- Turn

Some features are crucial when a pilot performs a turn: he must take the corner velocity into account. The corner velocity is a range of 330-340 KCAS airspeed at which the maximum allowable aircraft G and minimum turning radius can be generated. In other words, it is a range at which the aircraft turn performance is the highest.

- Acceleration

When you pilot your aircraft, accelerating is a basis. The purpose is to increase or decrease distance from an object. The **aircraft attitude** determines the effect of gravity on acceleration. If the aircraft velocity vector is above the horizon, acceleration effectiveness is reduced. If velocity vector is below the horizon effectiveness is enhanced. Moreover, the fastest airspeed gain occurs when you are to 0 G. **The speed you go** influences your acceleration rate. It is the best in the speed range from 300 to 400 KCAS. The altitude at which you fly impacts the acceleration. The higher the aircraft flies the more effective the acceleration is because of increased thrust.

2.1.2 Fight maneuvers

The purpose is to list some fight maneuvers and show when they are performed and why. Consequently this could be used for the bot to know when it has to perform a specific maneuver.

- Barrel roll

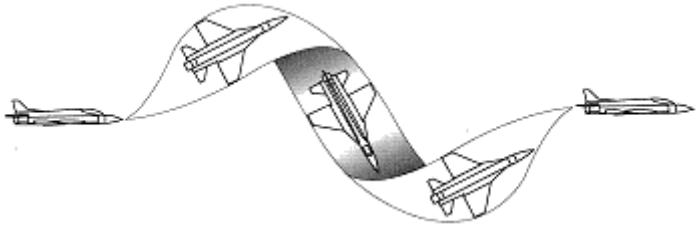


Figure 3: Barrel roll

The maneuver name comes from the flight path the aircraft performs as the aircraft rolls round a central axis. It is a speed-consuming maneuver. When a barrel roll is initiated, the aircraft starts performing a spiral. Therefore it increases the through air distance between the aircraft and the point to be reached. Because of the speed loss, it takes more time to go from one point to another one.

- Offensively it is used to prevent overshooting (because of speed loss)
- Defensively, it is used to cause overshooting (because of speed loss)

- Lead Turn

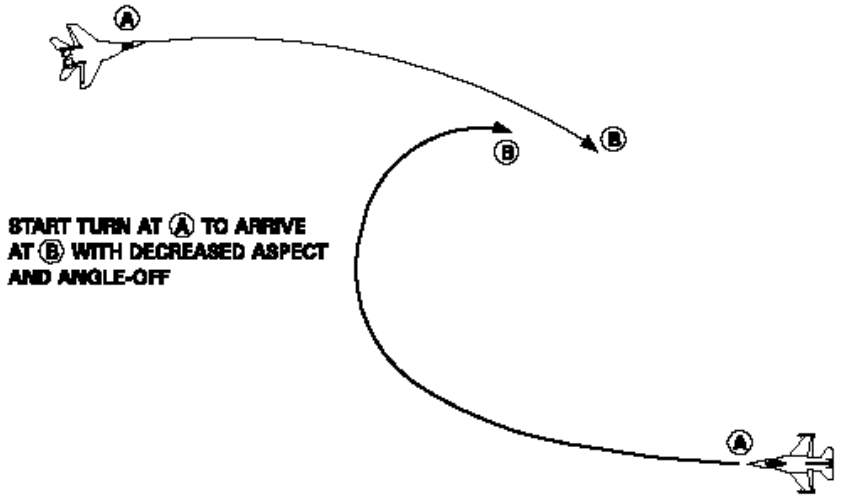
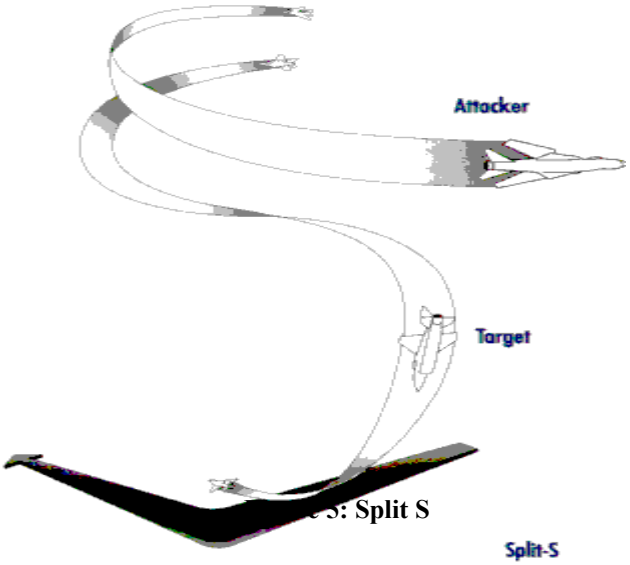


Figure 4: Lead turn

Two aircraft are facing. The lead turn aims at positioning the attacker in the opponent six 'o clock (the 6 'o clock of an aircraft is the rear if an aircraft). In other words, the attacker initiates a turn before passing the 3/9 line so that it can reduce the angle between its fuselage axis and the opponent's nose axis. It can be performed in any plane.

- Split-S

The Split-S is a diving half-loop. Therefore, it is used to increase speed or bleed off altitude. It is a high altitude maneuver. The Split-S is useful when you want to disengage a threat. Offensively, it is used to proceed a vertical lead turn (high to low).



: Split S

- Immelman

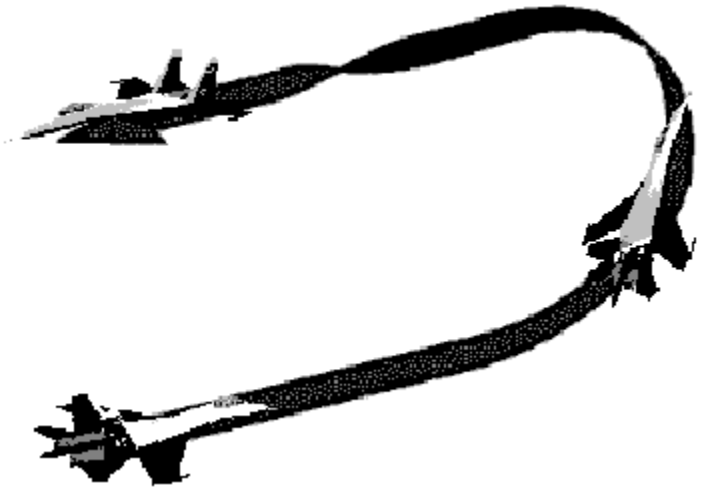


Figure 6: Immelmann

Immelman is neither an offensive nor a defensive maneuver. It looks like a half loop. It exchanges speed for altitude and reverses direction. Offensively, it can be used to proceed a vertical lead turn (low to high).

- Break Turn

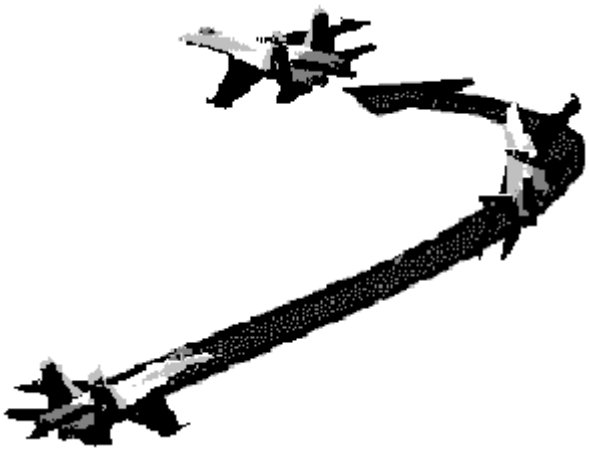


Figure 7: Break turn

The break turn results in a tight turn with high roll angle. It is a defensive maneuver used to avoid a missile or to deny the bandit weapon employment opportunity when he stands in your six. The point is to create as much angle as you can on the bandit. The break turn is high energy consuming.

2.2 About the pilot model

The point of this section is to provide some key information about the way a fighter pilot decides which maneuver he has to perform in a dogfight. These clues have been taken from a real F16 handbook downloaded from the Internet. In this document [Hand92], the pilot reasoning process has been analyzed. This analysis will be discussed from the following perspective: trying to extract some good rules of thumb about when to perform such a maneuver in order to automate such a decision making process.

2.2.1 Overview

A combat maneuver consists of positioning your aircraft so that the enemy is in your weapon parameters. But it is also vital to keep your enemy from employing his ordnance against you. Maneuvering your aircraft is the ultimate step of an iterative and complex process of reassessment: observing, predicting, and maneuvering. The final step is a kill or disengagement.

To execute theses tasks, a pilot is required to keep in mind some crucial information:

- Both aircraft characteristics
- Enemy maneuvers
- Energy state of both aircraft
- Its weapons range
- Pilot proficiency
- Both aircraft positions
- Maneuver risks

These parameters can be easily classified:

- Positional geometry (both aircraft position, angles, range)
- Weapons envelope (weapons range, angle required to shoot)
- Pilot Experience (Pilot proficiency, both aircraft abilities, abilities to assess the enemy's maneuvers, maneuver risks)

These notions must be defined before dealing with the attacker approach.

2.2.2 Vocabulary

2.2.2.1 Positional geometry

When speaking about an aircraft's position relative to another, a pilot deals with range, aspect angle and angle-off. A pilot uses these parameters to evaluate whether he has a positional advantage.

Range is the distance between two aircraft.

Aspect angle describes the relative position of the attacker to the target, without regard to the attacker's heading. It is defined as the angle measured from the tail of the target to the position of the attacker.

Angle-off is primarily concerned with the relative headings of two aircraft. Angle-off is defined as the angular distance between the longitudinal axes of the attacker and the defender. Whenever the attacker is pointing at the defender, the aspect angle and angle-off will be the same.

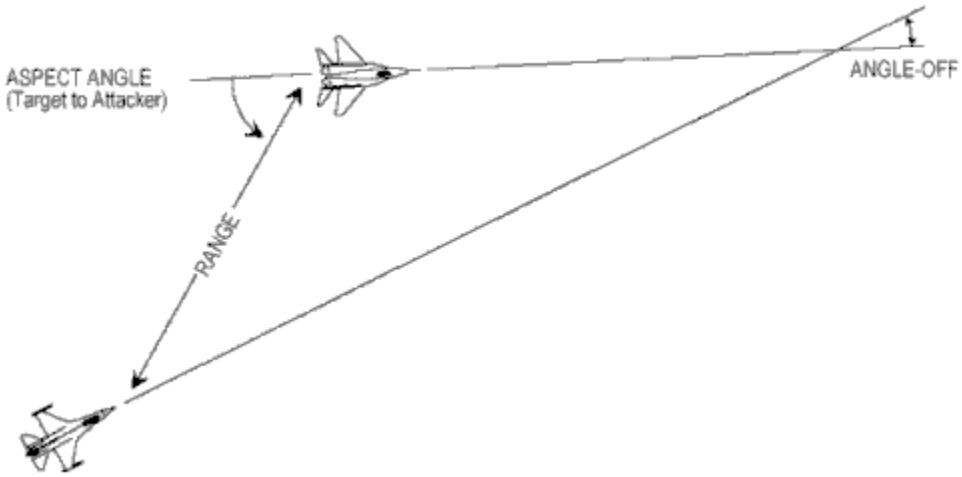


Figure 8: Position geometry

Generally speaking, it can be assumed that the closer you are from your target the better your chances are to shoot it down. The less angle problems there are (small aspect and angle off), the closer the attacker is to the ideal shooting position.

All the previous concepts can be merged to give more tactical oriented notion. A pilot visualizes the two aircraft separation in terms of **turning room**. Turning room is the offset or distance from the bandit. Turning can be acquired in either the lateral or vertical planes or a combination of both. In order to understand the concept of turning room, turning circle should be introduced.

Turning Circle is a concept involved in the approach maneuvers executed by an attacker. Turn circle is simply the defender's path that an attacker cuts through the sky when the defender turns. The first goal of an attacker is to enter into its opponent's turning circle. Once in the defender's turn circle, the attacker can initiate a direct pursuit to position into its opponent's vulnerable cone.

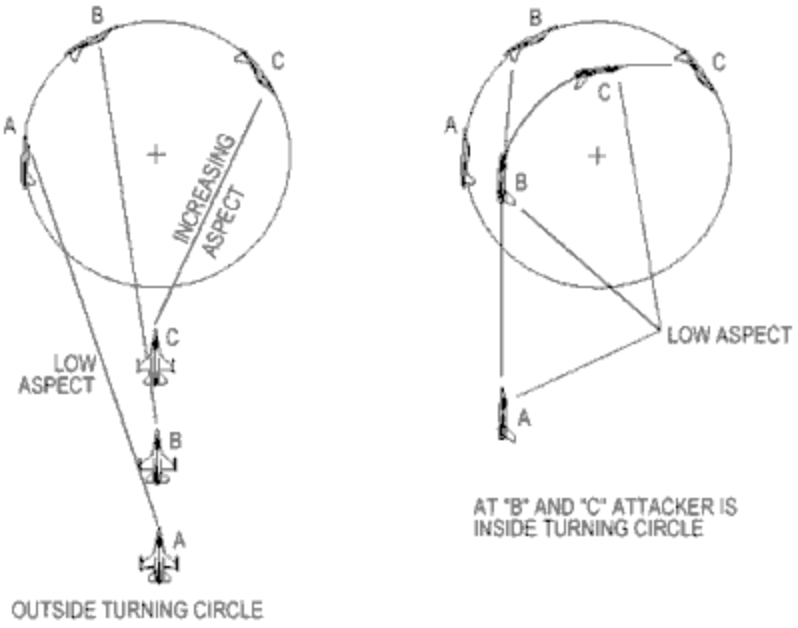


Figure 9: Turning circle

When tracking an opponent, an attacker aircraft's nose position can be in three different ways when its nose can point in front of the defender, closure rate increases, it is called **lead pursuit**. Behind the defender, closure rate decreases: this is **lag pursuit**. In both cases, angle-off increases. If it points at the adversary it is **pure pursuit**.

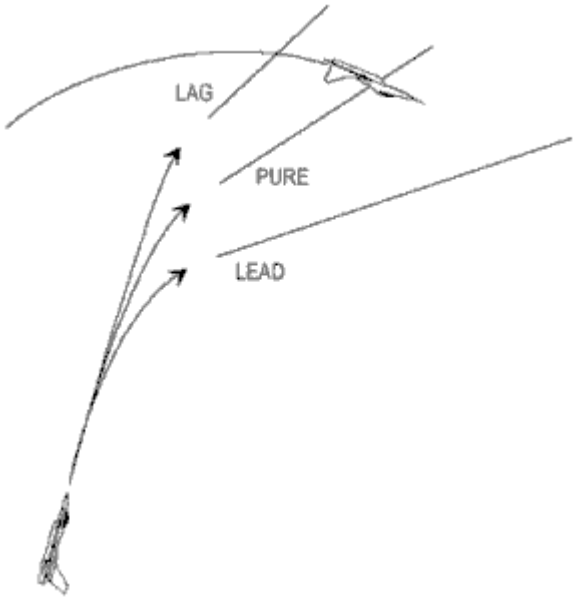


Figure 10: Attack pursuit courses

According to angles and range, a pilot elaborates its sequence of maneuvers. Reassessing the turning room he modifies it. The way the pilot takes these parameters into account is described in the section “Offensive approach”. Besides as the Artificial Dogfight player is required to be as close as possible to a real dogfight pilot, the pilot model will also use these criteria in order to choose a maneuver.

2.2.2.2 Weapon envelope

The **weapon envelope** is the area around the bandit where your missiles or gun can be effective. These weapon requirements impose a maximum range, aspect and angle-off and pursuit course. That defines the notion of vulnerable cone of the opponent.

The gun weapon envelope is a circle around the bandit depicting the gun's maximum range. There is no minimum range circle.

2.2.2.3 Pilot Experience - Energy and position principles

The pilot game plan is established by following a principle acquired through practice: the constant balance between energy and position advantage.

Energy is composed of kinetic energy (speed) and potential energy (altitude). Maneuvering consumes energy. When you have more energy than your opponent you have a serious **energy advantage** on him. When you are maneuvering, you consume energy and there is a constant trade off between speed and altitude.

Besides, the ultimate purpose of an attacker pilot is to position his aircraft in weapon parameters. When you are in your opponent’s six hours, you definitely have a **positional advantage**. A good pilot should ask this question. How much energy or future maneuvering potential can be expended for a given positional advantage? That is why a pilot must constantly balance energy and position.

As solution, fight maneuvers have been designed to achieve weapon parameters within the minimum energy expenditure in the minimum time as possible.

2.2.3 Offensive approach

In this part it will be shown how the pilot’s behavior during dogfight (especially with guns) can be logically analyzed and organized for later use.
Basically, there are two different situations when dog-fighting, which are shown in the following diagram:

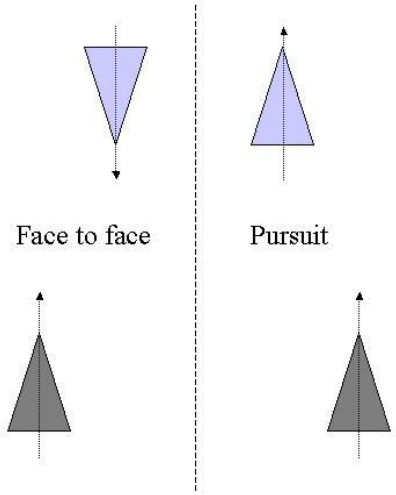


Figure 11: Dogfight situation

The project focuses on the pursuit situation seen from an attacker point of view. Consequently what follows only concerns the attacker.

2.2.3.1 Steps to shoot an enemy

Here follows the state description in which it is liable to be when trying to gun an enemy.

- Outside the Turn Circle: getting inside of it.

One of the main issues when dog fighting is entering the opponent's turn circle (TC). Actually, as long as you stay outside his TC, the aspect angle will increase and you will not be able to use ordnance. (see Figure 9)

- Inside the Turn Circle

The goal is now to solve the problem that has been created while transitioning TC: angles, range, speed etc. That is to say to place the aircraft in a position where ordnance can be employed. The maneuvers to be executed depend on what the bandit is doing (type of defense, maneuvers…). Mainly, there are two types of defense:

  - Check & extend: the enemy ends his turn and accelerates to increase range and his energy. Consequently the attacker may find himself outside the TC. Thus he should again enter the TC.
  - Continuous turn: the enemy wants to create angle problems, and bleed down the attacker's energy.

- Closing for guns

That represents a state when all is being done to place the aircraft in gun position. Closing for guns means also small range and speed. Two main strategies are provided when trying to get in firing position:

  - Pull lead: try to maintain a lead pursuit and gun the enemy.
  - Bid to lag: it consists in depleting enemy's energy before gunning him. The principle resides in floating back to the enemy's elbow while he is losing energy. The easiest way to proceed is to ease off of your turn to follow the enemy with a lag.

2.2.3.2 State machine

The previous phases can be incorporated in a state machine that stands for pilot states. Each state corresponds with a goal. For instance, when the pilot is "Outside the Turn Circle", his goal is to "get inside the Turn Circle". Considering this goal, information about the pilot, his aircraft and the enemy, and depending on the maneuvers he can proceed, he takes a decision to attempt this goal. The information taken into account will be discussed in the Chapter 1. The following picture illustrates this state machine, only considering the pilot as an attacker that wants to gun the enemy.

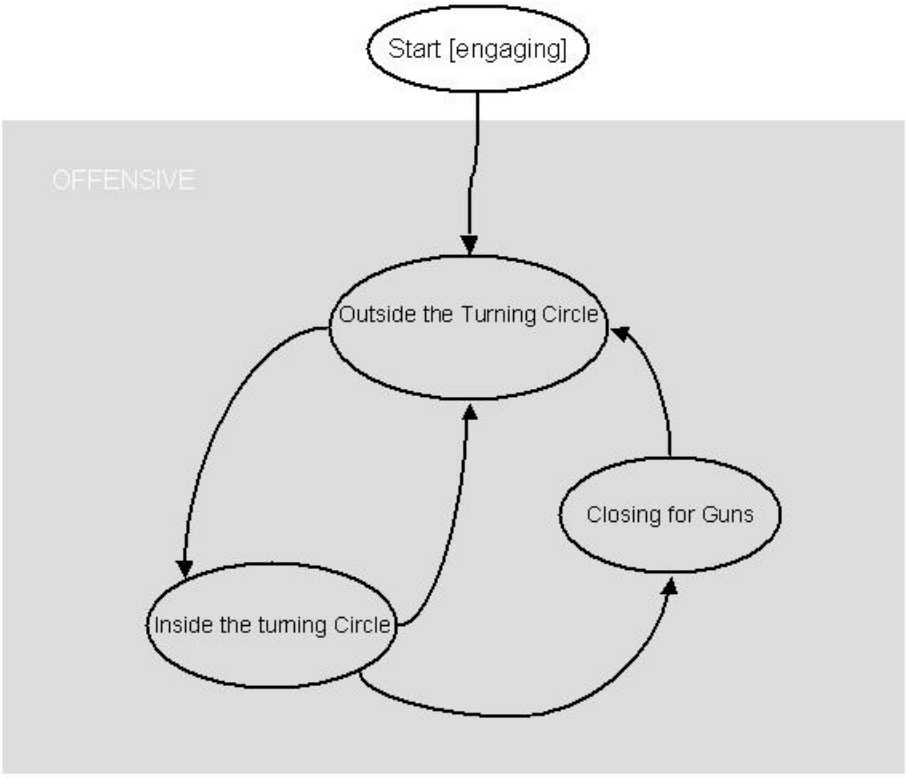


Figure 12: State machine

Chapter 3 Decision making model

3.1 Project approach

An iterative approach has been adopted. The main steps are the following:

- Collecting information
- Preliminary pilot model design
- First draft of the bot
- Final release of the bot

3.1.1 Collecting information

Developing a field specific artificial agent is a challenging goal for a computer scientist. An additional problem is that he is not necessarily an expert in a specific application domain. Thus the first step of our approach was acquiring the specific domain knowledge. It was necessary to know what the basis of dog fighting is, what the pilot maneuvers are. This collecting information task was time-consuming. It was discussed further in Chapter 1.

3.1.2 Preliminary pilot model design:

To claim developing a dog fighting artificial player, you need to create a model of the pilot's behaviour. More than '*what*' a pilot does, you need to know '*how*' and '*why*' a pilot acts this way: choosing the most accurate maneuver, what parameters should be taken into account etc. There are to approaches to extract knowledge:

- Discovering rules from data extracted from experiments
- Finding out rules from documents and literature

From the experiment point of view, we are limited. Information has essentially been collected from documentation. .

3.1.3 First draft of the bot

From the pilot model, a first software design draft could be carried on. It took into account few abilities an artificial pilot should have. A first implementation of the software was completed. The goals were:

- Detecting the serious defects and mistakes in the first pilot model.
- Implementing the basic functionalities of the bot.
- Being able to say what can be or not be done.

3.1.4 Final release of the bot

By taking into account the mistakes (software and design) a final design of the bot was provided. Conclusions have been made.

3.2 Rasmussen's decision making model

According to Rasmussen [Embrey], the decision making process implies three different levels of consciousness.

The knowledge-based behavior level: the individual is conscious of every step of the tasks he performs like a beginner. Considerable mental effort is required. Besides, in this mode, he is in charge of interpreting inputs, evaluating alternatives by weighing their pros and cons.

The skill-based behavior level refers to the performance of highly practiced tasks. No need for conscious monitoring. Basically, it means triggering procedures according to certain cues in an automatic manner. For instance, emergency procedures usually require skill-based behavior.

The rule-based behavior level is an intermediate level: the level of conscious control is intermediate. It processes the information using rules. Processing means choosing the procedures by observing and identifying. Rules have been learnt by practice for instance, or working with experienced workers, or by training.

The following diagram shows how to differentiate these three levels:

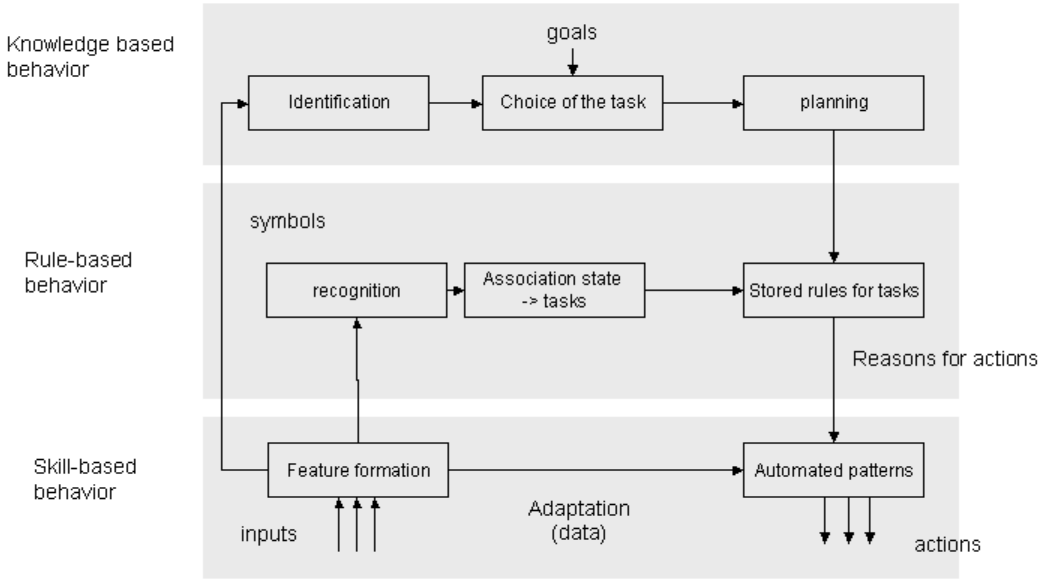


Figure 13: Levels of behavior (based on figure from [Embrey])

3.3 Decision making model

A name has been given to the model designed during this project. It is called 'Roger model'. To design this decision-making model, the main idea was to separate entities that perform intelligent tasks from those that perform automatic tasks.

The following diagram depicts these ideas.

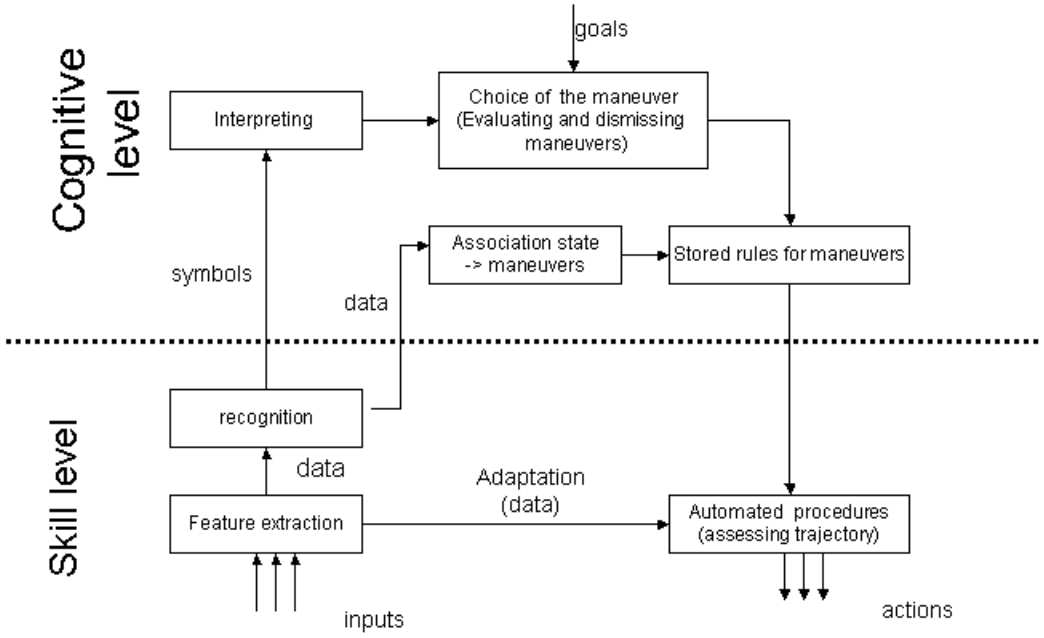


Figure 14: the Roger model

Two entities can be distinguished:

The **cognitive level** is in charge of intelligent tasks: that is to say that reasoning is required. Mental efforts are necessary so that the pilot workload is quite high. It is typically the resource used when he has a tactical choice to make. Given provided data of the environment, high-level goals are chosen. Basically it deals with interpreting, dismissing and electing actions by weighing the pros and the cons. For a dogfight pilot case, it means:

- **Interpreting the environment** represented in the form of symbols.
- **Electing a set of possible maneuvers** given the current situation
- **Evaluating the maneuvers** choosing one according to several criteria: incompatibilities ('increase the Turn Rate' is not compatible with 'increasing the Speed') and priorities.
- **Setting the rules** associated to the chosen maneuver.

The **skill level** has to perform the automatic tasks. It does not consume as many resources as the **cognitive level.** The way information is processed is very simple. Incoming information is directly processed to provide low-level orders. No decision is made. Basically for a dogfight pilot it is:

- **Recognizing the situation**. Rough data provided by the sensors are processed to get symbols ("the current position is good"). They are interpreted by the cognitive process.
- **Feedback controlling** to check if the current position is far from the expected one. If far, the pilot re-assesses the trajectory to fly.
- **Assessing the trajectory** to fly given the goals to reach.

3.4 Similarities and differences with Rasmussen model

Rasmussen model is based on **three levels of consciousness**. The lower the level is the less conscious monitoring is required. Our model is built on **two levels of intelligence**. The higher the level is the more it requires reasoning. Anyway they are some common points. The following diagram shows how both Rasmussen and our model diagram could be superposed.

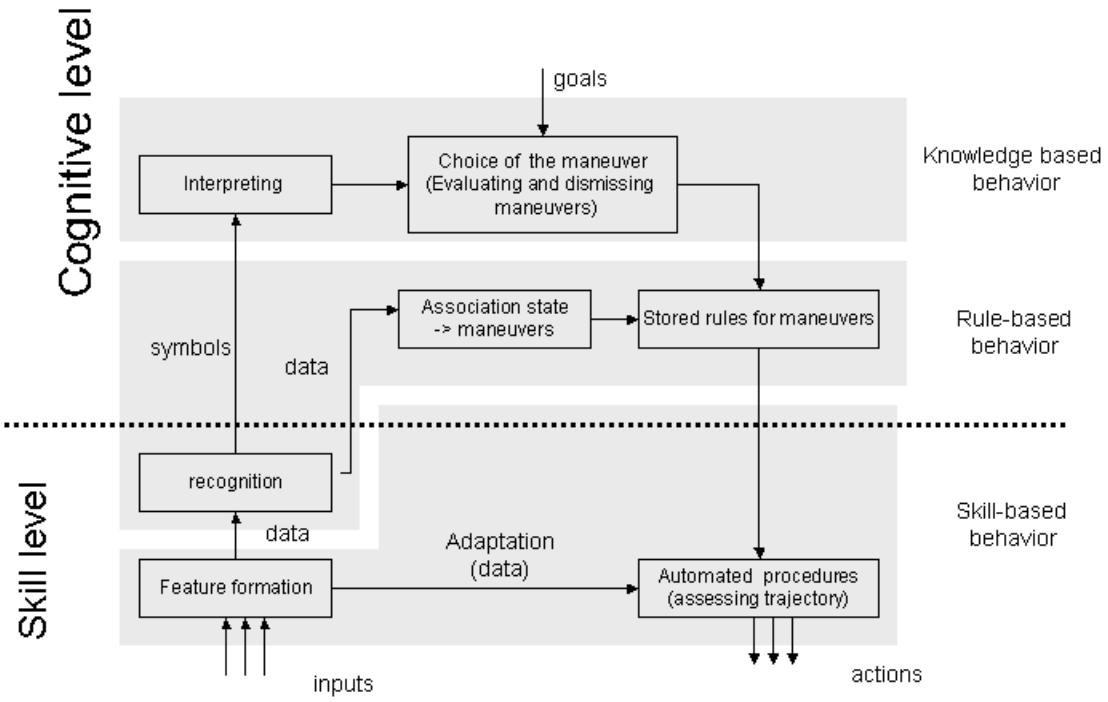


Figure 15: Rasmussen levels and our model on the same diagram

First the cognitive level stands mainly for the knowledge based behavior: a task that requires reasoning means that mental effort is needed. It should be carried out in a conscious manner. It is the case when an expert has to face a new situation. For a pilot it is:

- **Interpreting the symbols** representing the current situation
- **Choosing the most adequate maneuver**

Then, the Operational skills stand for the skill based knowledge. Dealing with automated tasks means performing highly practiced procedures without conscious monitoring. For a pilot it is:

- **Assessing trajectory.** Only the main features of the maneuver are required. The calculation of each point does not need conscious monitoring.
- **Updating the assessed trajectory** according to deviations

However, the rule-based level can be met on both mental cognitive and skilled-based process. A few functionalities belong to this intermediate conscious control level, for instance:

- **Recognizing:** this ability is acquired by practicing and interacting with experienced pilots. That is why it belongs to Know-how level
- **Finding a set of available maneuvers** given a current situation: it requires evaluating a decision tree to choose the set of possible maneuvers. That is why it belongs to the cognitive level.

To conclude, both models even if there are different can be met. For sure our model is simpler than Rasmussen's and less advanced notions are involved. Nevertheless, in what follows it will be often referred to Rasmussen levels because it might enable to explain things in a clearer way.

Chapter 4 First prototype

Before releasing the final release, a first prototype has been made. The purpose of this chapter is precising what has been included in the prototype. First some assumptions have been made. They are listed in the first section. Then, following the Rasmussen's model terms, all the different levels of the prototype are described in term of algorithms, principles and processes.

4.1 Assumptions

The first part deals with simplifications made on the **scenario** itself, that is to say, the 'characters' and their roles. The **knowledge** section is about what assumptions were made to address the missing information issue. The last part describes the simplifications for the **decision making process.**

4.1.1 Scenario

It is assumed that in a dogfight only two aircraft are involved. To remain simple, one is supposed to have an advantage on the other one. That is to say one aircraft tries to shoot another one: from now it is called the attacker. The other one tries either to reverse or to escape the situation. It is called the defender or the opponent. The bot only computes decisions for the attacker aircraft. Reversal situation are not supported. As soon as the attacker is almost overtaking the defender, that is to say, as the attacker is almost becoming the defender itself, geometry constraints are violated so that the program does not provide coherent outputs anymore.

4.1.2 Knowledge

As some **know-how skills are missing**, many abilities such as computing the acceleration to reach one point from another one have been implemented by setting data at random. For instance, the AI layer provides what it wishes via goals it wants to reach. It is up to the operational skilled-based process to find out how to reach these goals. The cognitive process provides the wanted *speed (t1)*. The know-how layer knows the initial *speed (t0*) and *speed (t1)*. The acceleration can be computed following this approximation.

$$Acceleration(t_0) = \frac{Speed(t_1) - Speed(t_0)}{t_1 - t_0}$$

4.1.3 Decision process

- **Input simplifications**

To compute decisions **only the positional aspects** of the inputs have been taken into consideration. For instance, only range and angles are processed to provide 'maneuvers'.

The same way as the geometry configuration, the **energy** state of both aircraft should influence the decision process of the bot. Let us consider the following situation: the defender is turning very close to his corner velocity. To be able to get to his six, it is necessary to turn at least as tight as him. If energy constraints are not taken into consideration, the decision to increase the turn rate will be taken. Turning at corner velocity consumes kinetic energy. Thus, if energy consideration

can influence the decision, the bot will not choose to tighten its turn but will prefer to wait until the enemy wastes energy.

It is supposed that **all information is accessible** to the bot at any moment. Right now, all information is required so that the conditions of the decision tree can be checked and the rules fired. In reality, it is not so obvious. The information the bot acquired is distributed over a time lap, so sometimes all data is not available for instance future predications.

- **Output simplifications**

The output maneuvers deal with **turns** only.

- **Decision making domain of validity:**

By 'validity' it is meant that there are some **constraints**. Beyond these limits, the behavior of the attacker is not necessarily coherent. The bot provides coherent outputs as long as these constraints are satisfied. First of all some of them are geometrical:

- First of all, the attacker has to be **in the defender's turn circle**. It is the first area around the defender an attacker should enter before getting into defender's six.
- The attacker should be **behind the enemy** (i.e. aspect angle within [-90,90]). The shadowed area in the diagrams below shows what is meant by "behind".
- The attacker should fly **the same direction as the enemy**. It means that, when the attacker is on the defender left side corner, it should point right, when it is on the defender right side corner, it should point left. The third figure shows a situation where the attacker does not point the right direction.

The first two figures below illustrate situations the prototype can deal with, whereas the third figure shows a forbidden situation (the attacker does not fly the right direction)

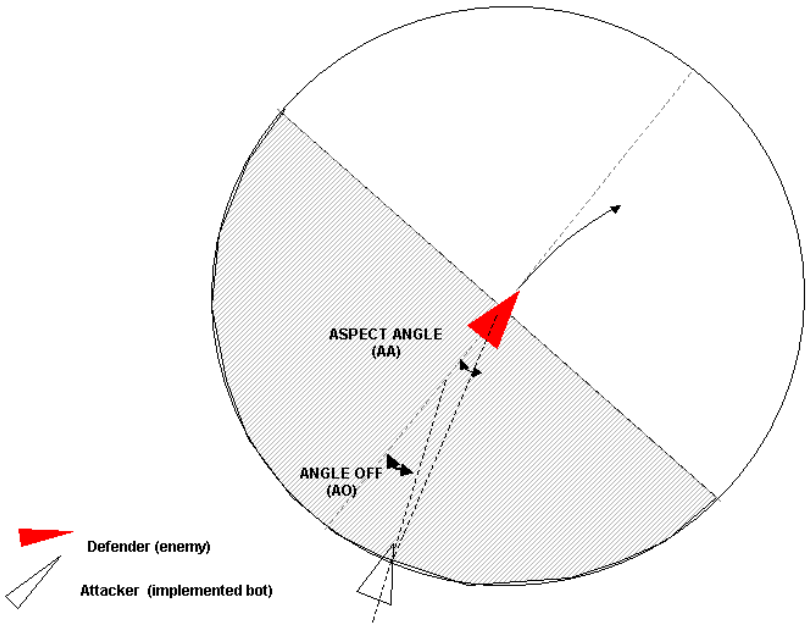


Figure 16: situation with a low aspect angle and angle off

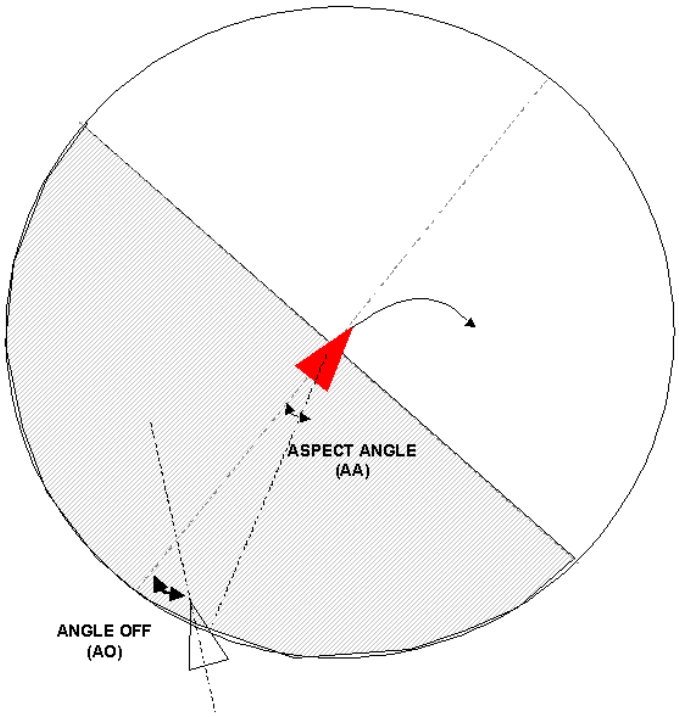


Figure 17: low aspect angle but high angle off

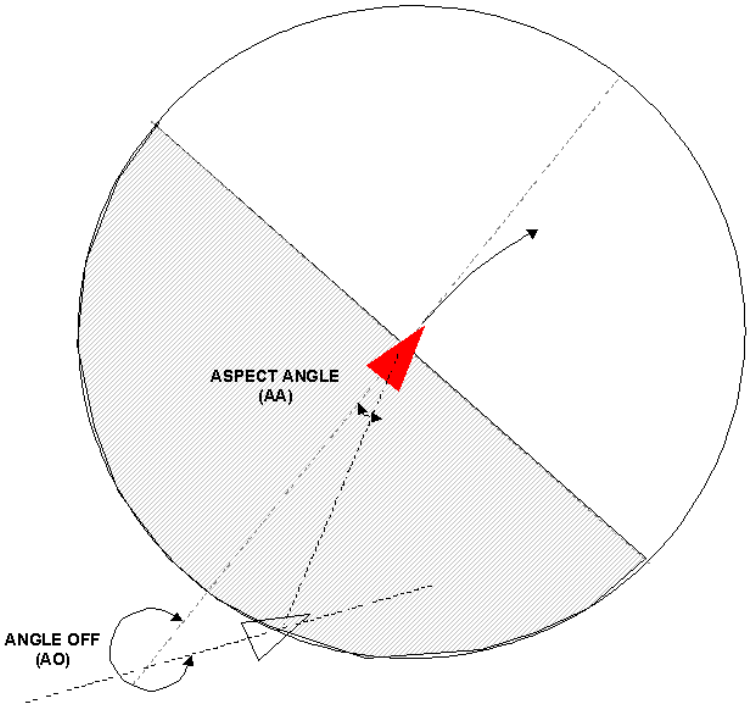


Figure 18: the attacker doe not fly the right direction

We assume that both aircraft fly **at a constant altitude**. Right now, neither inputs nor outputs take altitude into consideration.

- **Decision process simplification**

About the current prototype, the decision making process **does not take into account any predictions** on the enemy's trajectory. For example, let us suppose that at *t0* and *t1*, the AI module provides a maneuver. It is assumed that between *t0* and *t1* the enemy is still turning the same way. At *t0* there is no prediction about what will happen at *t1* except the fact that the enemy is turning with the same turn rate.

In the following sections, more details are given about the way the model has been carried out in the prototype. Since it is very close to the Rasmussen's behavior model, our model will be described using the following terms:

- Knowledge-based behavior
- Rule-based behavior
- Skill-based behavior

4.2 Knowledge-based level

In this part, the level of consciousness is high. The reasoning level is high too. The following diagram is taken from the figure of the paragraph 3.2

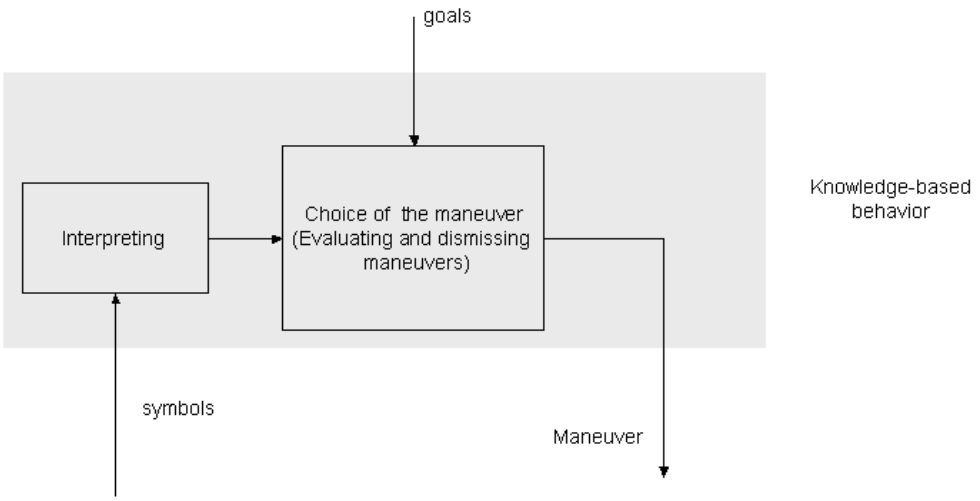


Figure 19: Knowledge-based behavior

In this part, it deals with interpreting the symbols and choosing the right maneuver.

4.2.1 Interpreting the symbols

Symbols stand for how satisfactory the bot considers the current situation. In this first draft, the used symbols only represent geometrical positions. They are based only on: angle off, aspect angle and range.

The symbols of this first prototype are the following:

SYMBOL	ASPECT ANGLE	ANGLE OFF	RANGE
GOOD POSITION	< LOW_AA	< LOW_AO	< TURNING_ROOM_LIMIT
MEDIUM AA POSITION	LOW_AA < *AA* < HIGH_AA	< LOW_AO	< TURNING_ROOM_LIMIT
MEDIUM AO POSITION	< LOW_AA	LOW_AO <AO <HIGH_AO	< TURNING_ROOM_LIMIT

The following diagram illustrates the meaning of these three symbols:

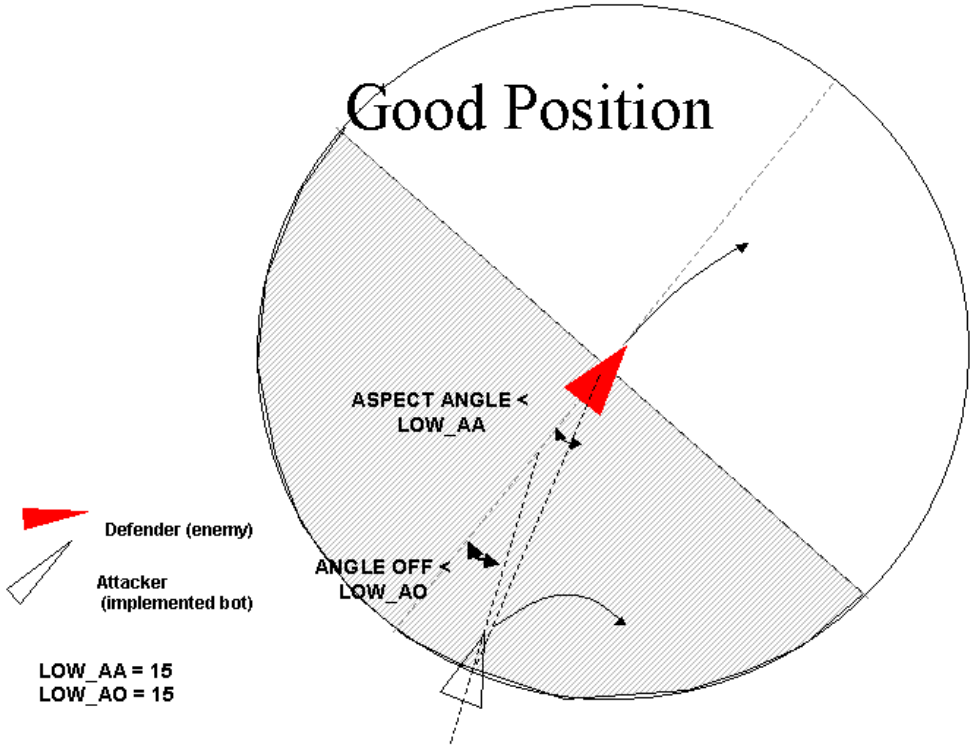


Figure 20: Good position symbol

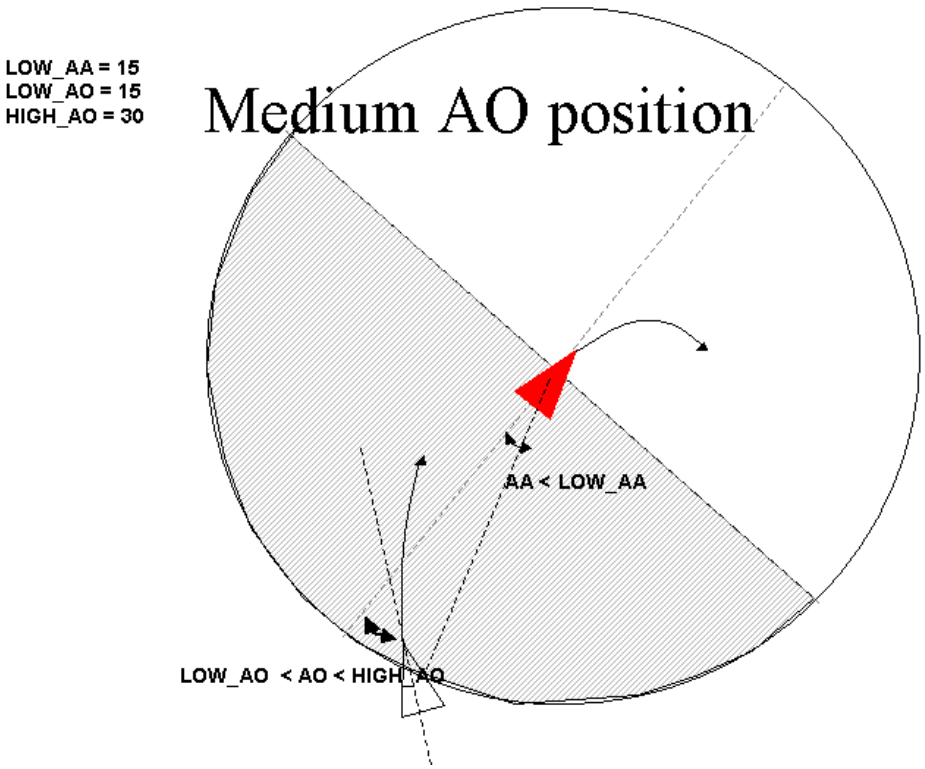


Figure 21: Medium AO position symbol

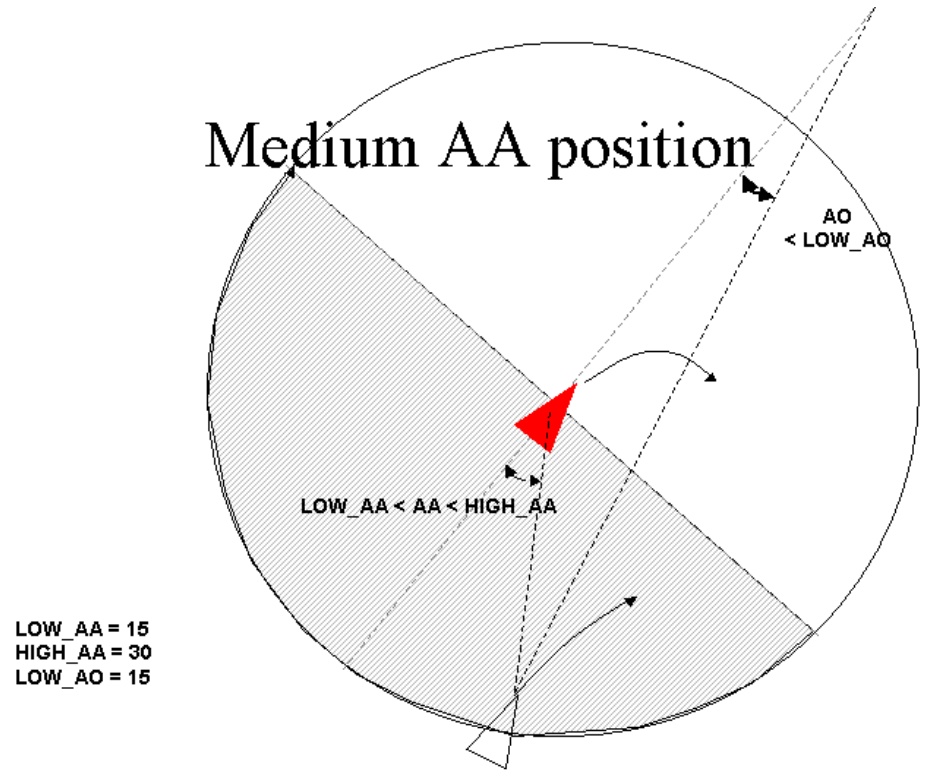


Figure 22: Medium AA position symbol

In a next release, it might be possible to add new symbols. For instance one might add symbols about energy states:

- **Low energy level**: both kinetic and potential energy are low.
- **Medium potential energy level**: loosing altitude can be dangerous.
- **Medium kinetic energy level**: the aircraft speed has decreased too much

Once a given situation has been matched with a given symbol the possible maneuvers are chosen. That is the functionality of the Rule-based behavior

4.2.2 Choosing the right maneuver

Normally thanks to symbols and the current situation data, it is possible to choose the most appropriate maneuver among the set of possible ones. For instance, if the symbol Medium AO position is associated with the current situation, maneuvers increasing the angle off will not be chosen.

Unfortunately this has not yet been incorporated in this first prototype.

4.3 Rule-based level

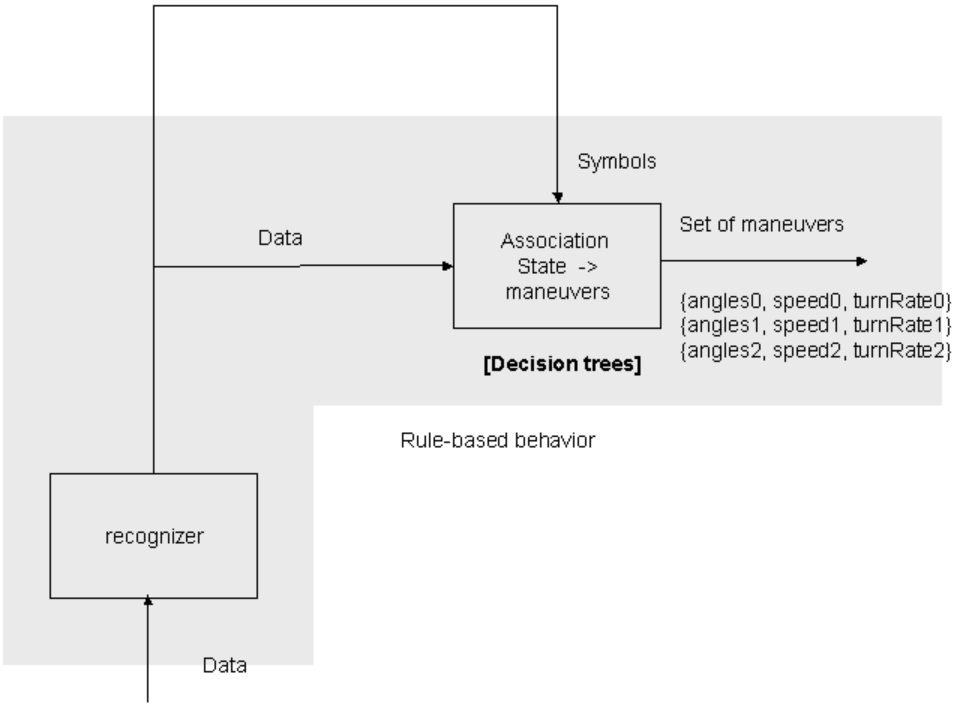


Figure 23: Rule-based behavior

4.3.1 Why rules and decision trees?

The choice of the possible maneuvers depends on two things: the **situation data** and the **symbols** associated with the current situation. 'Basic maneuvers' are 'pursuit' (or 'angles'), 'speed', and 'turnRate'. According to the F16 handbook, these parameters are the basis of the maneuvers performed by both attacking and defending turning aircraft.

A basic maneuver is designed to achieve a goal. It also describes how to achieve it.

- **Angle basic maneuver**: it is a simple maneuver to address angle issues. It can be lead, lag or pure pursuit.
- **Speed basic maneuver:** it is the kind a basic maneuver to address closure rate problems. It can consist of increasing, decreasing, or maintaining the throttle.
- **Turn rate basic maneuver:** it is a basic maneuver to solve turn rate problems. For instance turning tighter than the enemy.

First it deals with finding out the maneuvers that are candidates to be executed. That is to say finding the candidates {angle, speed turnRate} by processing symbols and data. According to the F16 handbook, once a situation has been provided and the pilot has a mental representation of it, a basic maneuver can automatically be associated, in a few words data and symbols determine a set of possible basic maneuvers. Naturally rules and decision trees are the most natural way to represent this rule of thumb.

As there are three main basic maneuvers to select, there are three decision trees in the decision process.

An example:

```
SYMBOLS, DATA -> {ANGLES, SPEED, TURNRATE}, {ANGLES, SPEED, TURNRATE}..
```

In the prototype, due to the decision trees only one set of parameters {angle, speed, turn Rate} are computed. That is to say only one maneuver is selected so there is no need to dismiss maneuvers.

```
SYMBOLS, DATA -> {ANGLES, SPEED, TURNRATE}
```

4.3.2 How have the decision trees been built?

There are three decision trees, one for each basic maneuver. The first level of the tree combines symbols with the current situation. The lower level enables to know what basic maneuver should be chosen. Each node is associated with a condition that has to be satisfied in order to enter a node. A leaf is associated with a basic maneuver.

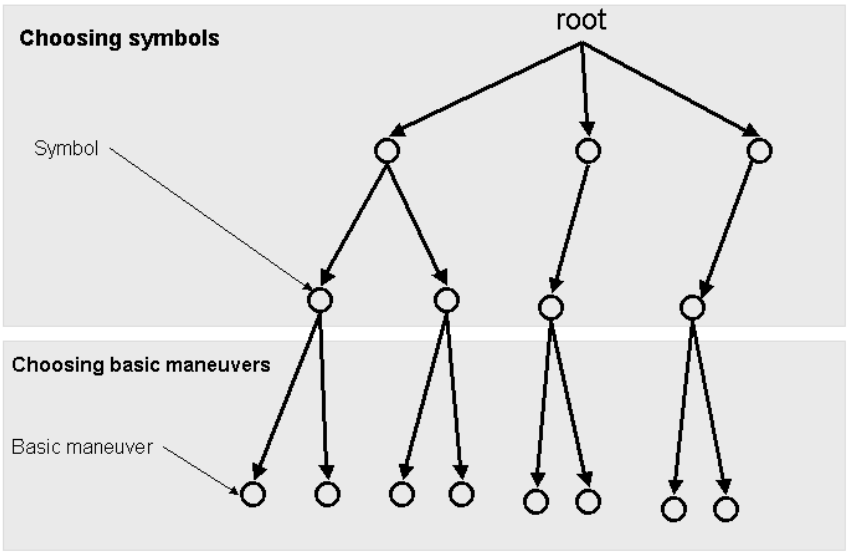


Figure 24: Design of decision tree

Here is an example taken from the decision tree 'angles':

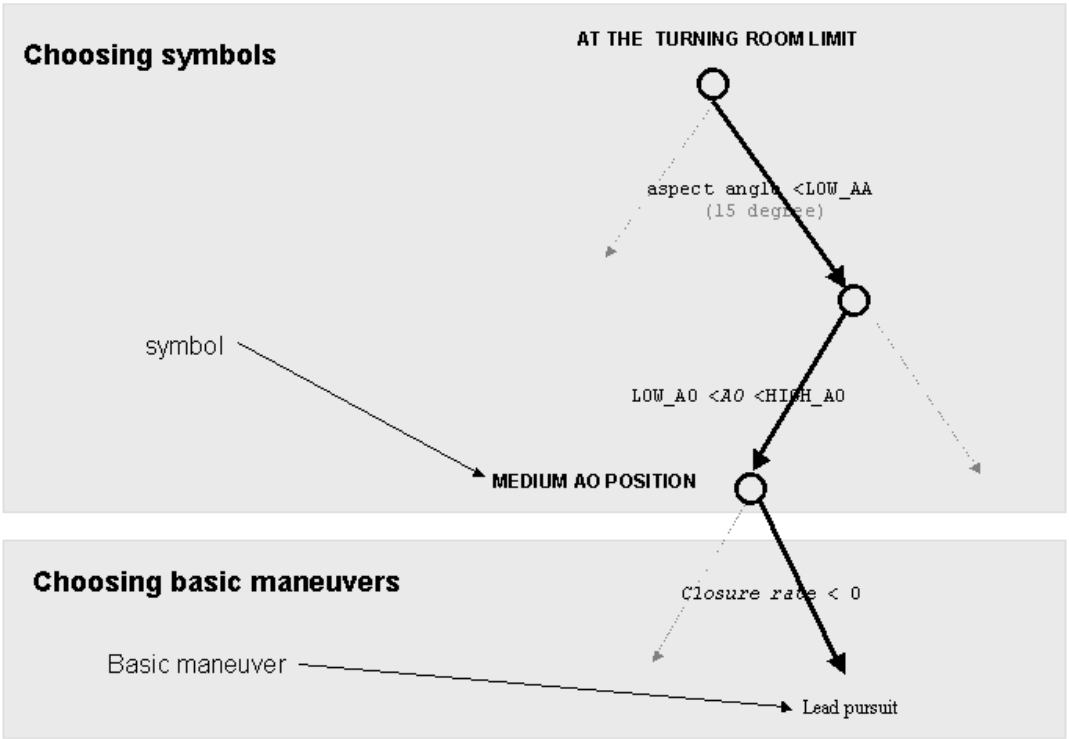


Figure 25: Example of a decision tree

The branch shown above has been taken from the pursuit tree. It shows how the decision process is run.

- First, if the aspect angle is low enough, the corresponding node is entered
- Next, the angle off is checked, the 'medium AO position' node is entered.
- Then the closure rate condition is checked.

In the ultimate step, the leaf is reached. Constraints have to be set on angles so that the aircraft will be in a lead pursuit position. Actually flying in lead pursuit enables the attacker to solve an excess of angle off.

4.4 Skill-based level

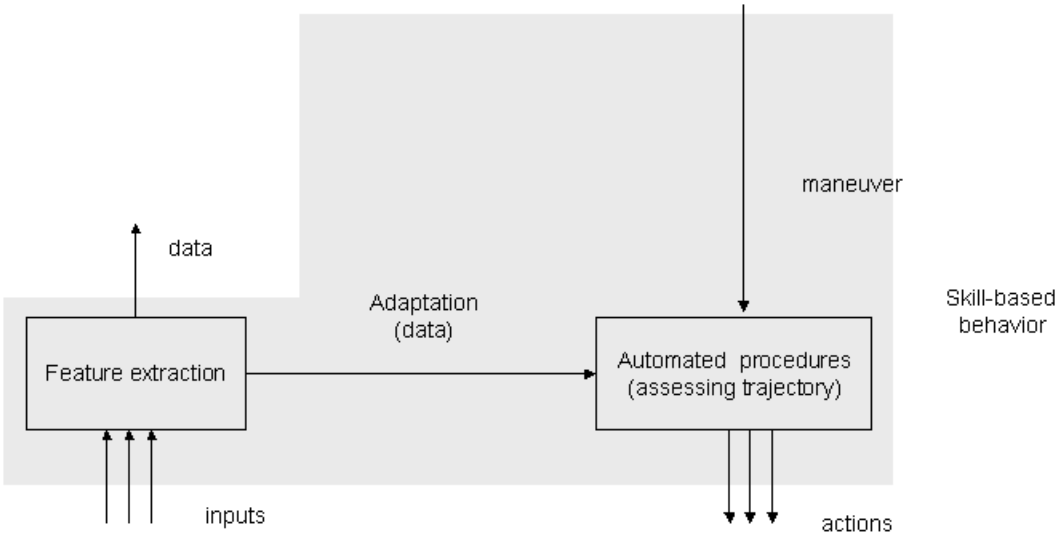


Figure 26: Skill-based behavior

The skill-based contains the information that is the most difficult to obtain. It deals with

- All **the automated procedures** a pilot has to execute when a maneuvers is to be performed.
- The **feedback control** performed by the pilot to check if his aircraft flies far from what he expected.

None of these functionalities have been implemented in this first draft.

The next diagram depicts this prototype.

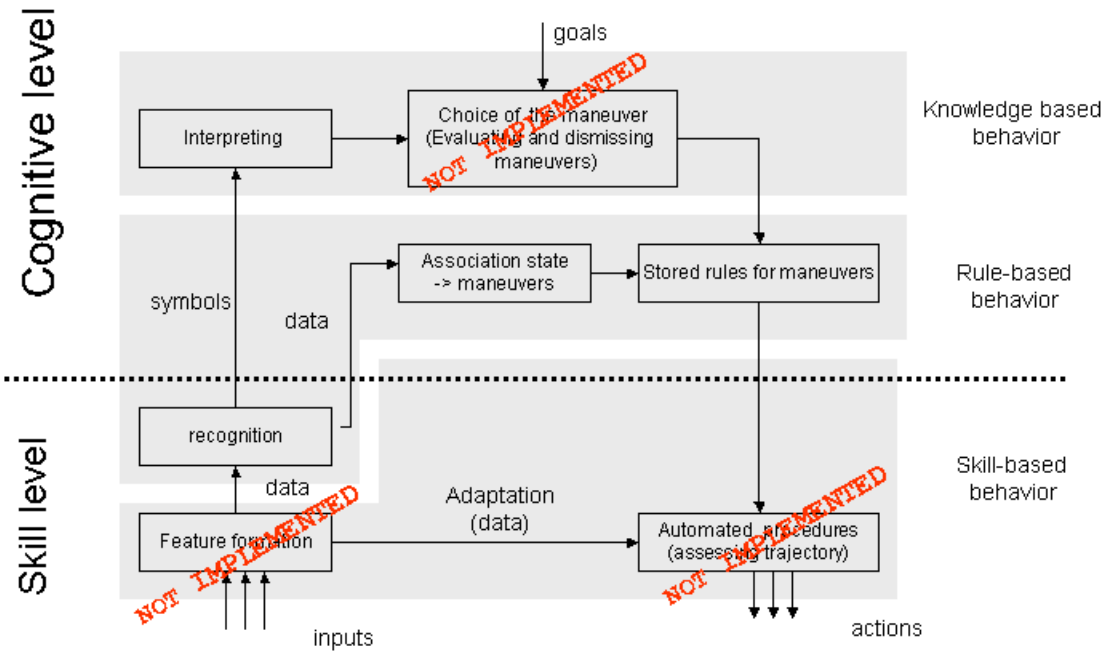


Figure 27: State of the art of implemented modules

4.5 Limitations

The goal is to design a bot. The first demonstration is a proof of concept. If the results are successful, this provides a basis for a second demo, which should include more human knowledge.

This first prototype is submitted to two kinds of limitations. First it is a prototype. It should be as simple as possible, the goals are essentially detecting the defects of the first pilot model, implementing the basic functionalities of the bot, being able to say what can be or not be done. It should not be too time-consuming.

Next the second kind of limitations is more general to the designing model, so it will also be valid for the final release of this bot: the pilot reasoning is a highly complicated process. It is very difficult for the model to have as many abilities as a real pilot. A lot of information is missing because we do not have access to all the data we need. some of them, concerning the F16 for instance are confidential and others require interviewing the pilots themselves.

4.6 Results

In a few words, testing the prototype consists of running the code of both the cognitive and skill-based process with a simulator engine. Data representing the behavior of the bot have been logged so that it is possible to compare what was expected and what was done in the form of diagrams and graphics. The first section describes the crucial data that are used in the program and that represent the behavior of the bot. Next, Graphs and their interpretations are provided. Finally, the main mistakes of the first draft will be discussed in the last section.

4.6.1 Overview of the prototype implementation

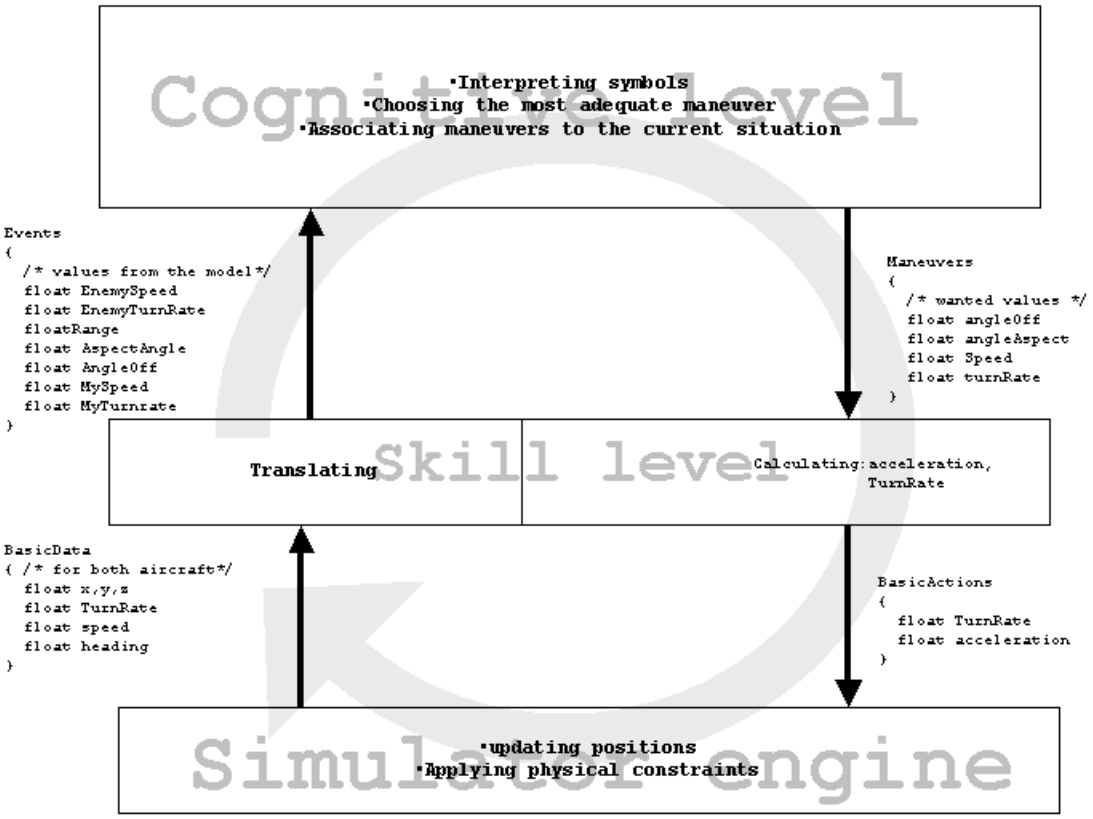


Figure 28: prototype implementation overview

This diagram is about the program that runs the prototype code. Both upper layers represent respectively the cognitive and the skilled-based processes of our model. It is the prototype. The lowest layer represents the flight simulator engine. It is not relevant to detail what is inside each of these layers. Nevertheless, analyzing the data exchanged by these layers will enable us to trace the behavior of the bot.

- **Events**

The cognitive process gets information in the form of Events. Events are high-level data. The bot that has to generate decisions can directly use them. They are designed to be used to generate decisions only; they do not require to be processed by any situation recognizer. In the prototype case, only the position aspects have to be taken into account. They are very basic:

```
EVENT
{
       FLOAT ANGLEOFF; // ENEMYHEADING – MYHEADING
       FLOAT ASPECTANGLE; // RELATIVE ANGULAR POSITION
       FLOAT RANGE;
       FLOAT ENEMYSPEED;
       FLOAT MYSPEED;
       FLOAT ENEMYTURNRATE;
       FLOAT MYTURNRATE;
       FLOAT MYG; // ACCELERATION OF MY AIRCRAFT IN 'G'
}
```

- **Maneuvers**

Decision making determines maneuvers that have to be executed. These maneuvers are high-level actions that do not require decision anymore. Maneuvers need only know-how skills to be processed. In the prototype case, only the position aspects have to be taken into account. They are very basic:

```
MANEUVER
{
    FLOAT TARGETSPEED; // WANTED SPEED
    FLOAT TARGETASPECTANGLE; // WANTED ASPECT ANGLE
    FLOAT TARGETANGLEOFF; // WANTED ANGLE OFF
    FLOAT TARGETTURNRATE; // WANTED TURNRATE
}
```

- **Basic Data**

It is the input of the bot. It stands for the data that are about to be processed by the artificial dogfight player. It represents the current state of the environment where the artificial player flies. Typically, they are the data that characterize both aircraft. The Flight simulator engine constantly updates these data.

```
BASICDATA
{
    FLOAT MYX, MYY, MYZ;
    FLOAT MYSPEED, MYTURNRATE;
    FLOAT MYHEADING;

    FLOAT ENEMYX, ENEMYY, ENEMYZ;
    FLOAT ENEMYSPEED, ENEMYTURNRATE;
    FLOAT ENEMYHEADING;
}
```

- **Basic Action**

Basic action is the output of the bot. It represents the orders computed by the artificial dogfight player. It is not guarantied that these intentions will be fulfilled. It gives all the requested information about the intended aircraft route so that the model can update the aircraft parameters.

```
BASICACTION
{
    FLOAT ACCELERATION;
    FLOAT TURNRATE;
}
```

4.6.2 Test results of the first prototype

As the bot is designed to position in the six ‘o clock of the defender, it is obvious to compare the defender’s data to the bot’s one. Such comparisons were possible by logging the basic data structures (inputs of the bot) and plotting them.

4.6.3 First graphs and analyses

Here are some diagrams made during the execution of the program. The initial situation was the following: the defender is flying a pretty constant curved trajectory. Its turn Rate is just a little bit changed during the simulation from –5.75 to –6.75 deg/sec. The defender is turning left. Its speed is constant: 370 knots.

- **Speed**

Y-axis: speed in knots
X-axis: time in seconds

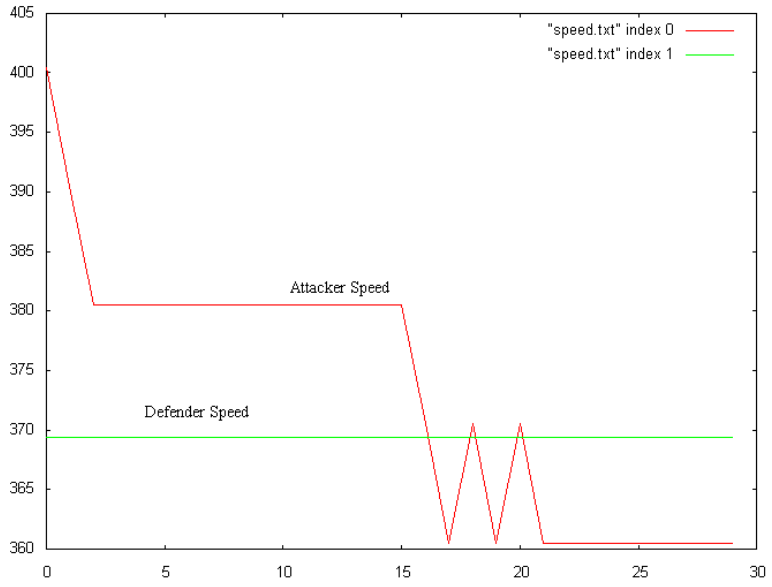


Figure 29: Attacker and defender speed

One can notice that the attacker’s speed gets closer the defender’s speed after a certain delay.

- **Turn Rate**

Y axis : turn rate in deg/sec
X axis : time in seconds

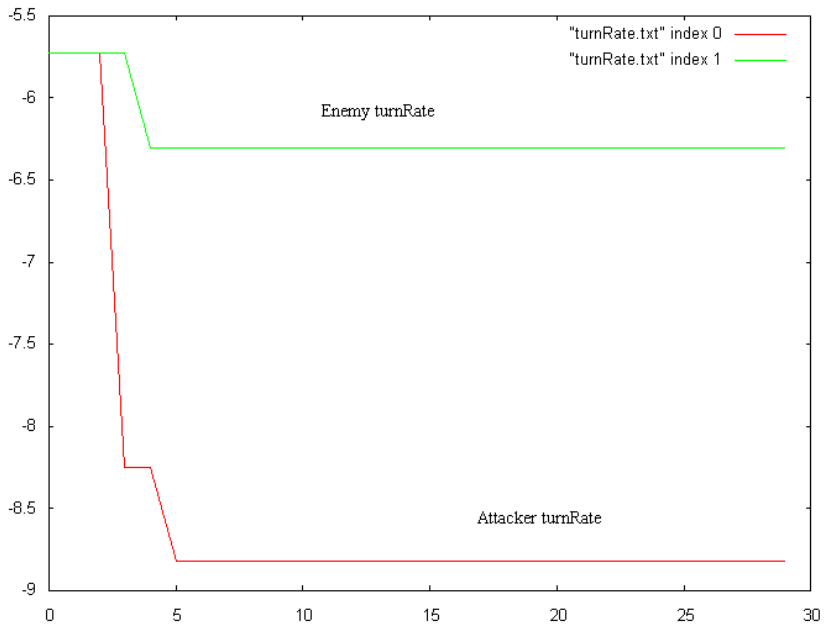


Figure 30: attacker and defender turn rate

On can notice that the attacker turnRate is higher that the defender one.

- **Aspect angle**

Y axis : aspect angle in degrees
X axis : time in seconds

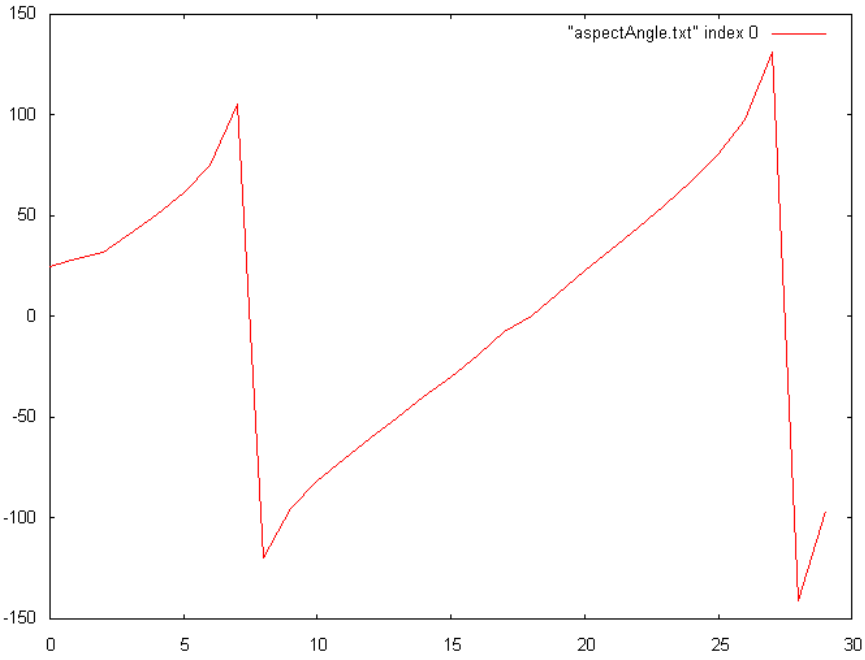


Figure 31: aspect angle

The aspect angles keep increasing. It is important to notice that the aspect angle range is [-180, 180] degrees.

- **Range**

Y-axis: distance between both aircraft in nautical miles
X-axis: time in seconds

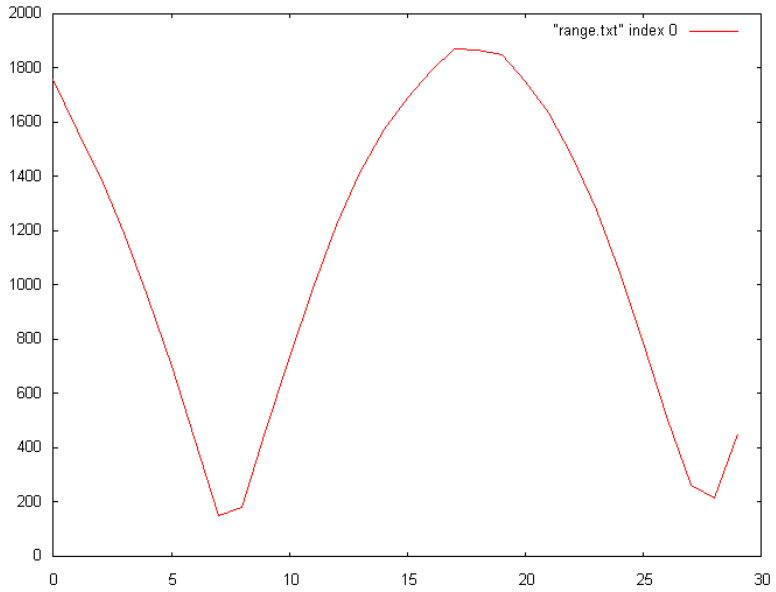


Figure 32: range

One can notice that the range is fluctuating a lot.

- **Analysis**

We notice first that the bot behavior is not satisfactory considering the aspect angle and the range graphs. None of the decisions correct the diverging values of aspect angles and range. Next, the pilot acts correctly concerning the speed. The computed turn Rate is not adequate because it should have fluctuated enough to correct the aspect angle issue.

Next, let us consider the bot performances. There is a reaction delay: it is pretty difficult to make a relevant correction. In fact, it is easier to check if a maneuver is correct on a qualitative point of view but it remains hard to tell if it is correct on a quantitative point of view. This is due to the lack of expert knowledge.

It is suspected that the maneuvers are not always adequate to the current situation. The following paragraph deals with further tests about that.

4.6.4 Further tests

The next tests are designed to show why the maneuvers may not be appropriate to the current situation. They consist of comparing 'events' and 'maneuvers' data at two different times of the program execution: *t =0s and t=3s*

1. ***t = 0 s***

Let's consider the current event datum:

```
>>>>>EVENT<<<<<<
ANGLEOFF: 12.60507
ASPECTANGLE: 24.75943
ENEMYSPEED: 369.36002
MYSPEED: 400.464
ENEMYTURNRATE: -5.729578
MYTURNRATE: -5.729578
MYG: 2.1020408
RANGE: 1755.9491
```

The computed maneuver is:

```
>>>>MANEUVER<<<<
>> 380.464 KNOTS (SPEED)
>> 15.0 DEGREES (AA)
>> 30.0 DEGREES (A0)
>> -5.729578 DEGREES/SEC (TURNRATE)
```

Let us detail for instance how the decision is taken concerning the angle aspect. Here is the branch that has been executed during the tree evaluation. It shows exactly what conditions have been satisfied. On can notice that the decision has been correctly made

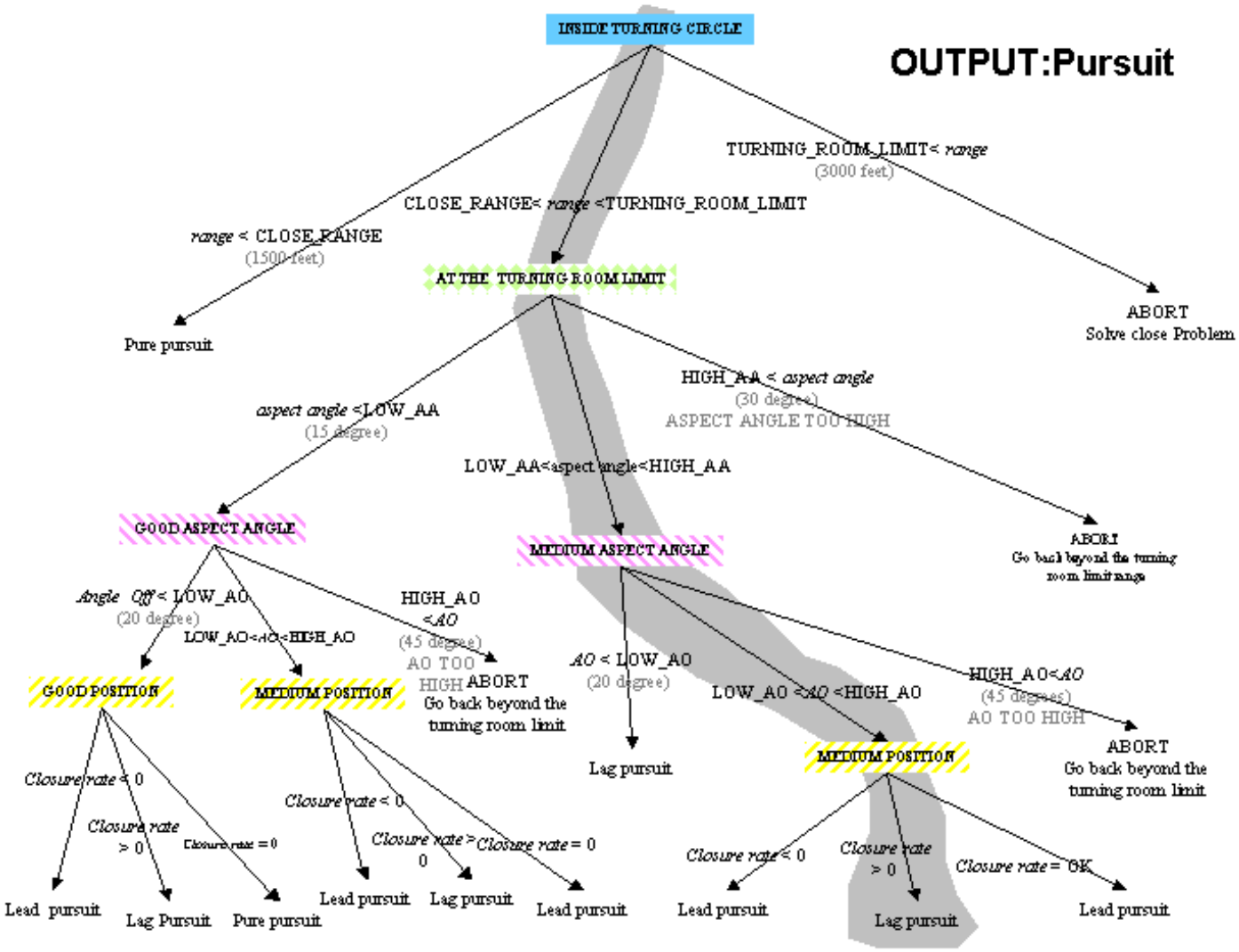


Figure 33: pursuit tree for t = 0 s

2. *t = 3 s*

The event datum is the following:

```
>>>>>EVENT<<<<<<
ANGLEOFF: 12.60507
ASPECTANGLE: 31.545
ENEMYSPEED: 369.36002
MYSPEED: 380.47968
ENEMYTURNRATE: -5.729578
MYTURNRATE: -5.729578
MYG: 1.9971429
RANGE: 1395.7313
```

The corresponding maneuver is:

```
>>>>MANEUVER<<<<
ABORT
NO SPEED COMMAND
>> 1.5 DEGREES (AA)
>> 3.0 DEGREES (A0)
NO TURNRATE COMMAND
```

Let us visualize what branch of the pursuit decision tree has been executed:

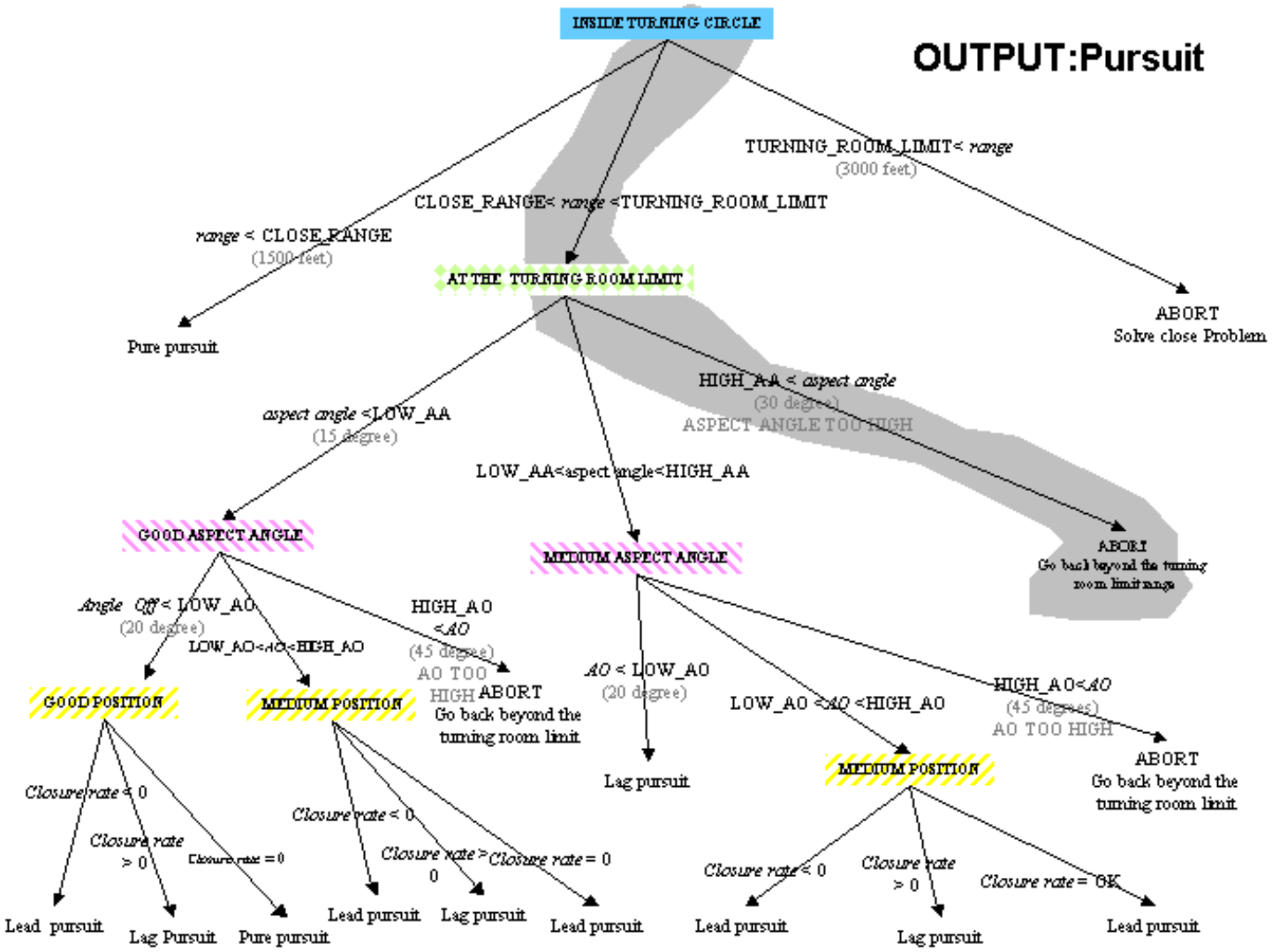
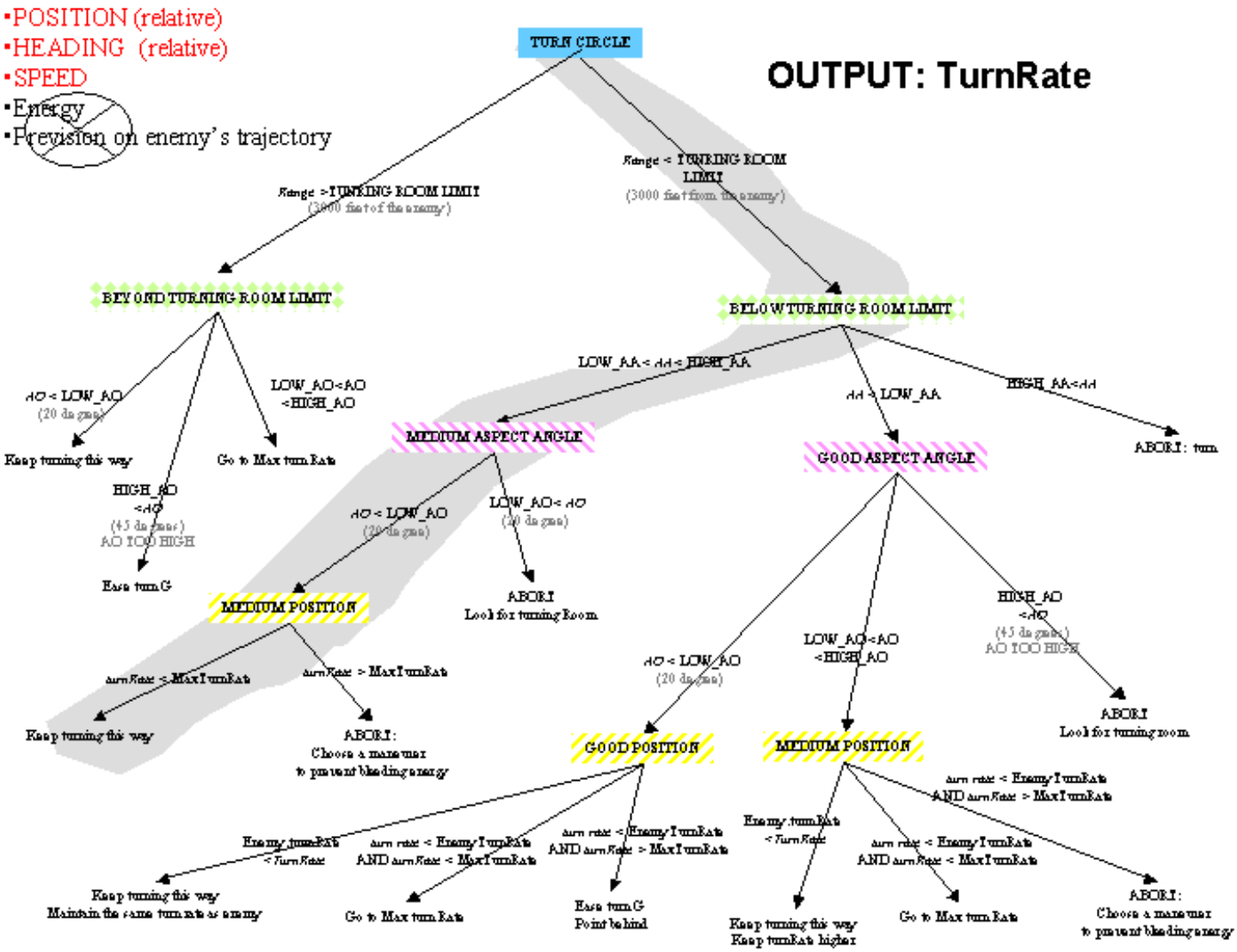


Figure 34: pursuit tree for t = 3s

In fact, the aspect angle was not corrected. From 24 degrees in the first sample it increased to 31 degrees in the third one and went beyond the maximum. That is why the 'abort maneuver' was fired. Even if the aspect angle was increasing, the decision concerning the turn rate remained the same: 'Keep turning this way' in the TurnRate tree. The system did not take any adequate decisions to make this aspect angle decrease.

The following diagram shows how the decision has been made



Each maneuver is composed of three basic maneuvers: one about the turn rate, about the angle configuration and about the speed. A detail has been missed in this first draft: these three basic maneuvers are not on the same level. Actually a maneuver is both a set of goals and methods to achieve these goals. For instance, the turn rate constraint is intended to be respected at the end of the certain time: it is a goal. In the other hand, the angle basic maneuver is more a method that should be applied during this time. Thus the TurnRate and angle basic maneuver should not be processed the same way by the know-how layer.

A goal should only be achieved within a certain time, however. On the other hand, a method tells how actions should be performed.

In a few words, in this first draw the main problems are:

- Basic maneuvers should achieve either 'goals' or/and 'methods' (or 'policies' both term will be used)

- A time period should be defined so that a goal should be achieved at the end of it. Moreover, during this period the pilot should use the computed methods. This notion of period does not exist in the prototype. All basic maneuvers were translated into actions that were executed immediately.

Given these crucial issues, no more tests have been carried out, because they would not have been relevant. Correcting these problems will be the point of the final release.

Chapter 5 Final release

5.1 Assumptions

Basically, the assumptions are the same as in the prototype. Moreover, many problems came from the assumptions made on the knowledge part of the bot. Thus, the simplifications have been modified and adjusted to get more satisfactory results.

5.1.1 Knowledge

The fact that some expert skills were missing was the crucial issue of the first draft. Many assumptions have been improved. We still do not have expert skills but behavior has been set that seems coherent for anyone that is not an expert.

5.1.1.1 Trajectory generator

Once the cognitive level provides a maneuver, the skilled-based level has to compute the trajectory the bot should fly. This feature was not implemented in the first draft, because the notion of *future trajectory* did not really exist. The trajectory calculator has to compute aircraft parameters (speed, angles etc) every *t* seconds within the prediction period. These parameters should respect the constraints dictated by the maneuver (lead pursuit, increasing the speed…etc). Physical equations are needed to compute such trajectory parameters. As there might be very complicated basic equations have been used. The main parameters that are calculated are angles, speed, turn Rate, and G force. Basically the following equations have been used to implement those algorithms:

$$TurnRate = \frac{g}{v}$$

$$turnRadius = \frac{v^2}{g}$$

Where *g* is the value of the G force applied to the aircraft and *v* its speed.

Thanks to these relations, once two parameters are known it is quite easy to compute the last ones.

5.1.1.2 Feedback controller.

The feedback controller is in charge with comparing the expected location to the current real one. The distance between both is assessed. It is not only a geographical distance but also a distance between aircraft parameters that represent its state (angles, speed). *This state distance* has been measured very basically. Some default values have been set to enable such comparison: *LOW, MEDIUM* and *HIGH* so that the distance between the expected angle configuration and the current one may be considered as *MEDIUM* if at least the difference between the expected angle off and the current angle off is *MEDIUM* for instance.

5.1.1.3 The incompatibility problem

The analysis of the current situation can lead to several possible maneuvers. Some of them can be incompatible that means that some of their main features can be incompatible. To simplify the incompatibility issue, the incompatibilities between features have been stored in the memory of the bot. There is no more reasoning anymore. For instance *lead pursuit* (pointing in front of the defender) is clearly incompatible with *decrease turn Rate*. The bot does not search for why it is not compatible. It just knows it.

5.2 Knowledge-based level

The knowledge-based level should perform the tasks that need a high level of consciousness. For instance interpreting the current situation by using symbols. Next, it can select the most appropriate maneuver. The Figure 19 illustrates this process.

5.2.1 Interpreting the current situation

The used symbols are the same as in the prototype except two symbols that have been added. They represent the situation when the cognitive level does not have to process any decision.

SYMBOL	ASPECT ANGLE	ANGLE OFF	RANGE
GOOD POSITION	< LOW_AA	< LOW_AO	< TURNING_ROOM_LIMIT
MEDIUM AA POSITION	LOW_AA < *AA* < HIGH_AA	< LOW_AO	< TURNING_ROOM_LIMIT
MEDIUM AO POSITION	< LOW_AA	LOW_AO <AO <HIGH_AO	< TURNING_ROOM_LIMIT
BAD POSITION	HIGH_AA<	X	< TURNING_ROOM_LIMIT
BAD POSITION	LOW_AA< AA< HIGH_AA	LOW_AO<AO	< TURNING_ROOM_LIMIT
BAD_POSITION	AA<LOW_POSITION	HIGH_AO< AO	< TURNING_ROOM_LIMIT

5.2.2 Choosing the right maneuver

In the first draft, only one maneuver was computed. In the final release several candidates are computed. The knowledge level is in charge with selecting the most appropriate one. Let us see what “most appropriate maneuver” means.

A maneuver is designed to enable the aircraft to achieve goals within the prediction time: improving angle position or solving closure rate problems, solving turn rate and G force issues. These goals have priorities that are changing with the current situation. A maneuver should:

- Achieve as many goals as possible within the prediction time.
- Respect goal priorities.
- Be coherent

In fact we can notice that a basic maneuver (see 4.3.1) corresponds to a goal.

- Angle basic maneuver -> angle
- Speed basic maneuver -> closure rate
- G, speed basic maneuver -> turn rate
- G basic maneuver -> G

Goals also represent parameter values of the final state that should be reached within the end of the prediction period. No attention is paid to how these values are reached. It is not the role of the Knowledge-based level to check whether these goals will be reached. That is the skill-level duty (see section 5.1.1.2). To each goal there is a set of candidate 'basic maneuvers '. Basic maneuvers represent *policy*, that is to say recommendations about how to reach this final state. Here are the 'goals' and the possible corresponding 'basic maneuvers':

- Angle -> lead, lag, or pure pursuit
- Closure rate -> increase, keep, decrease speed
- Turn rate -> increase, keep, decrease turn rate.
- G -> increase, keep, decrease G forces.

A goal (and thus all corresponding 'basic maneuvers') has a priority. It is assumed that the goal priorities are fixed:

1. Angle
2. Closure rate
3. Turn rate
4. G force

Incompatible basic maneuvers are stored in the bot memory. For instance 'increase speed' is incompatible with 'decrease Turn rate'.

Actually, there is no selection of "the most appropriate maneuver", the idea is to only build the most appropriate maneuver. To perform such a task, an algorithm computes a maneuver that should:

- Be composed of coherent basic maneuvers
- Respect the goal priorities.

To do that, during the evaluation of the decision trees all-incompatible encountered basic maneuvers are dismissed. The last ones are sorted according to priorities so that a maneuver is obtained.

```
BUILD MANEUVER (DECISIONTREE [] TREES) : MANEUVER []
{
      BASICMANEUVER [3][] CANDIDATES;
      BASICMANEUVER [3] COMPATIBLES;
      BASICMANEUVER [3] RESULTS;

      //THERE IS THREE DECISION TREES: PURSUIT, CLOSURE RATE, TURN RATE.

      FOR(I=0;I<3; I++)

            //EVALUATING EACH DECISION TREE AND RETURNING THE SET
            //OF CANDIDATE BASIC MANEUVERS, ONE SET FOR ANGLES,
            //ONE FOR CLOSURE RATE ..ETC

            CANDIDATES[I] = CHOOSE_CANDIDATE_BASIC_MANEUVERS (TREES[I]);

      // SELECTING 3 COMPATIBLE CANDIDATES
      COMPATIBLES = SELECT_COMPATIBLE_CANDIDATES (CANDIDATES);

      // SORTING THE COMPATIBLE CANDIDATES ACCORDING TO PRIORITIES
```

```
        RESULTS = SORT_CANDIDATES (COMPATIBLES);

        RETURN NEW MANEUVER(RESULTS);
}
```

The fact that *goals* and *policies* have been clearly separated is the biggest improvement of the new release decision-making process. The following diagram sums up how it worked without this separation and how it works with the distinction.

For the prototype, a maneuver was executed immediately. The goals had to be reached immediately although there were not designed to.

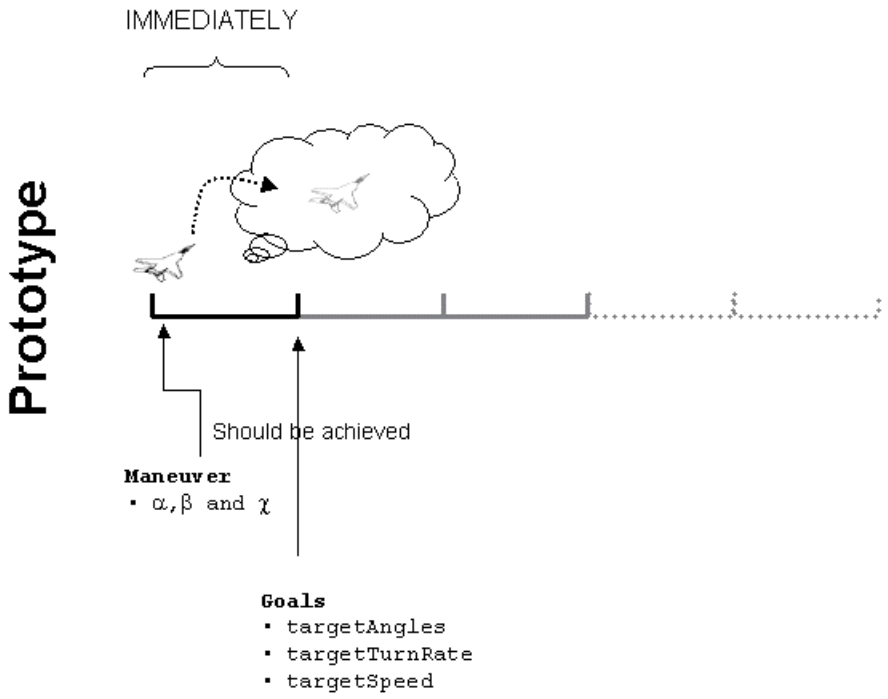


Figure 35: goals in the prototype

In the final release, a maneuver is composed of *goals* and *policies* so that it tells what state and how to reach it. Regularly the actual position is compared to the sub goal position thanks to distant function (see section 5.1.1.2) so that the trajectory is recomputed as often as needed. Thus, the *policies* influence the behavior of the aircraft during all the prediction period long.

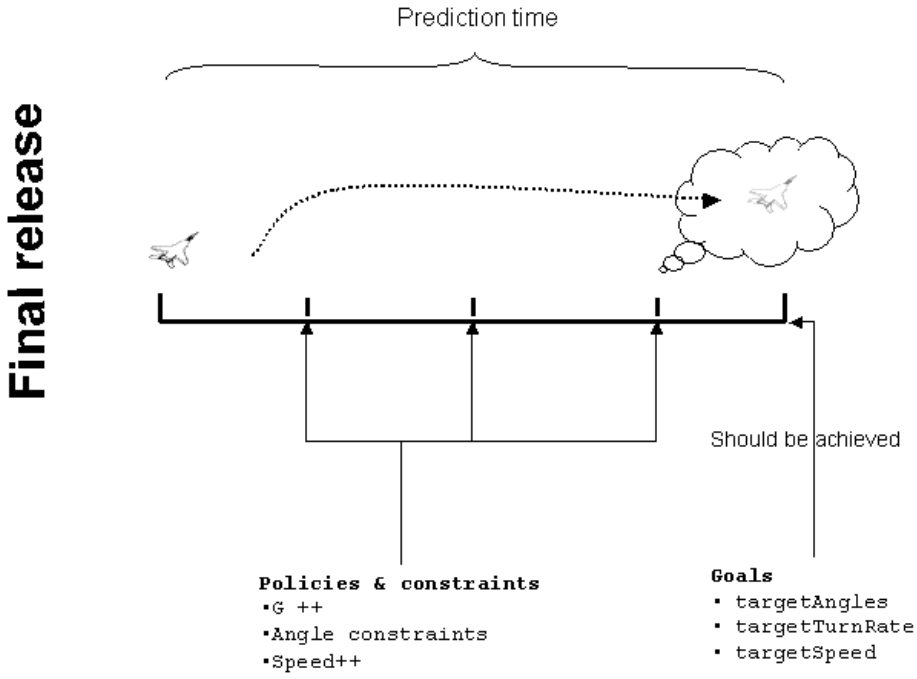


Figure 36: goals and policies in the final release

5.3 Rule-based level

The rule-based level is in charge with making the rough data from the flight simulator understandable to the knowledge-based level. From the Figure 23, we can see that the situation awareness module associates symbols to the current situation first. Symbols and rough data are processed and possible maneuvers are computed. To perform such a task, decision trees have been used. The way they have been designed is discussed in the following section. In section 5.3.2, further details are given about the situation awareness module.

5.3.1 Design of decision trees

Keep in mind that each decision tree is designed to compute one set of candidate basic maneuvers to achieve their corresponding goals. By running the decision process, the following steps are followed:

1. **Selecting symbols:** they should be associated with the current situation.
2. Symbols and data are processed **to give goals**.
3. **Basic maneuvers** that are adequate to the current situation are computed.

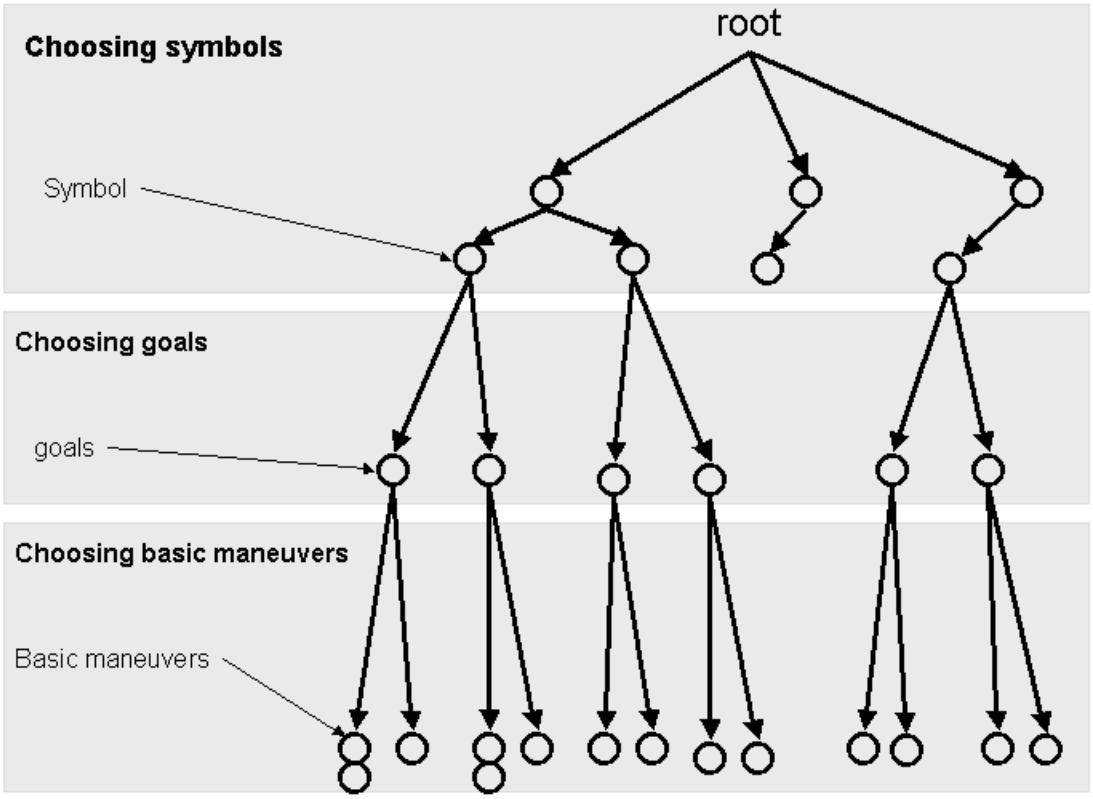


Figure 37: decision tree design in the final release

The first level of the tree permits to know what symbols can be associated with the current situation. The intermediate level enables to know what goals should be achieved. Each node is associated with a condition that has to be satisfied in order to enter a node. A leaf is associated with a set of candidate basic maneuvers

To be more concrete, in the decision making process; there is one 'choosing symbols' tree. It corresponds to the tree used by the situation awareness module. It is more detailed in the following section. Next for each symbol node, there are three sub-trees that are evaluated in parallel, one per kind of goal: pursuit, turn Rate, closure rate. The following diagram illustrates this statement:

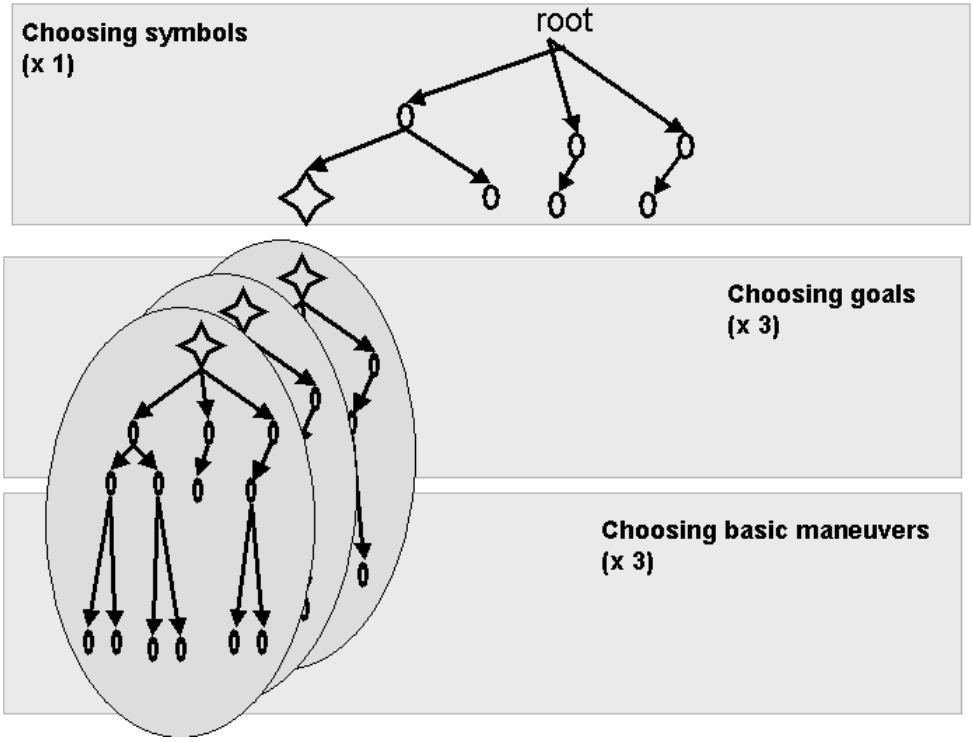


Figure 38: three decision trees per goal

Here is an example taken from the decision tree turn rate.

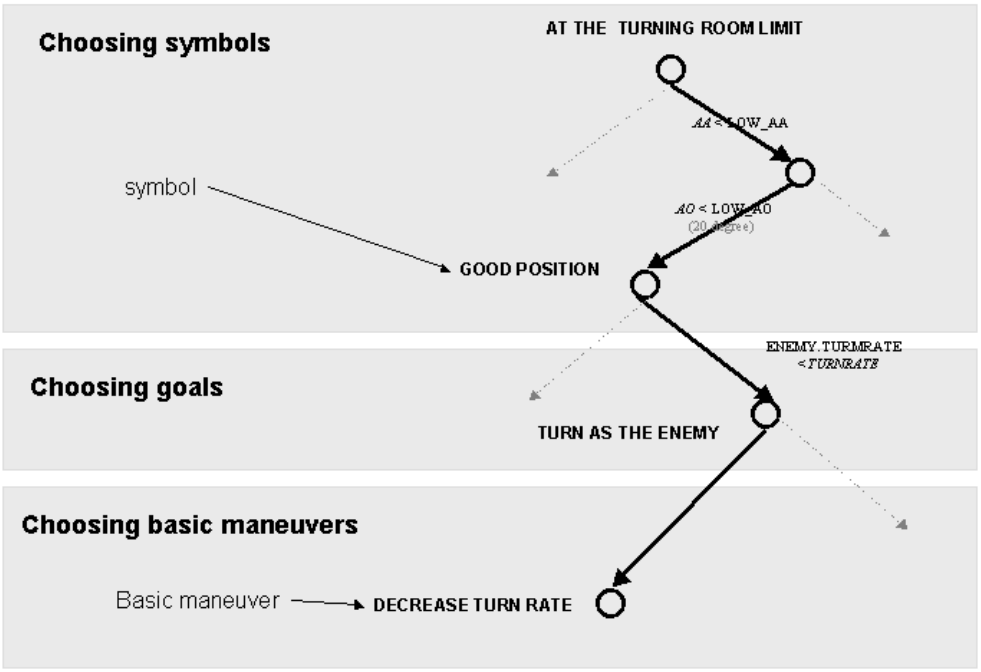


Figure 39: example of a decision tree

In the figure above, the branch is taken from the turn rate tree. It shows how the decision process is run.

- First, if the aspect angle is low enough, the corresponding node is entered
- Next, the angle off is checked, the 'good position' node is entered.
- Then the turn rate condition is checked.
- Finally, the basic maneuver 'decrease turn rate ' is reached.

What is called event is a set of data that characterize a situation. The data are not only quantitative but also qualitative. Consequently the event production consists of several recognizing processes that produce qualitative data from quantitative ones.

5.3.2 Situation awareness

A bot needs to have its own representation of the environment. The situation awareness module is in charge with associating symbols to the current situation. In addition to the positional symbols that have been listed in section 4.2.1, there are also energetic symbols. They represent how the pilot from an energy point of view, can interpret the current situation. Both of them constitute what is called an Event.

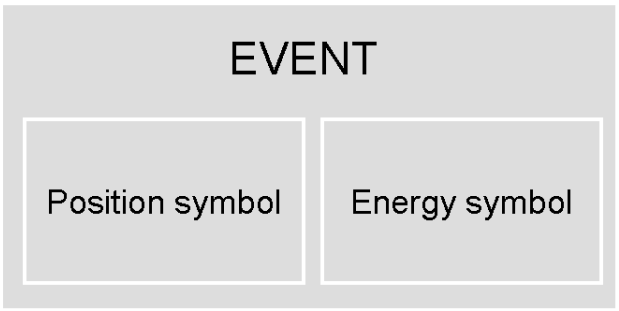


Figure 40: event structure

5.3.2.1 Recognizing position

From the rough data (also called 'formatted data') positional symbols are computed by evaluating the position recognizing tree. Depending on the rough data values different leaves can be reached. Each leaf represents a positional symbol. The following diagram shows the tree used by the position recognizer:

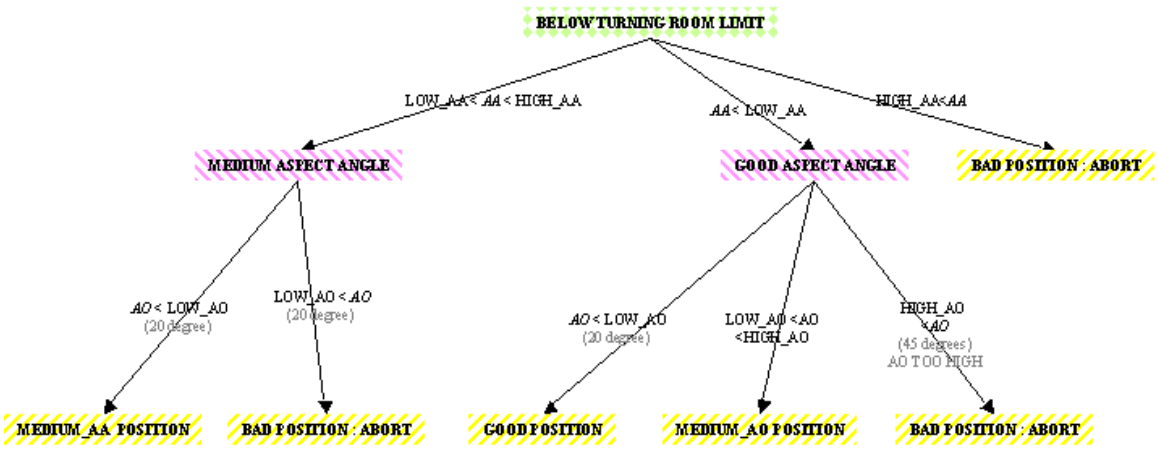


Figure 41: Position recognizing tree

5.3.2.2 Recognizing the energy state

The principle is the same to recognize energy state as to recognize position. The leaves of the tree inform if there is a potential danger or not. This part has not been incorporated in the final release.

5.3.2.3 Computing Event

1. Qualitative data are received. These rough data are information about aircraft speeds, turn-rates and relative positions. They are data form the flight simulator that have been formatted so that the bot can easily process them. These data are called FormattedData.
2. The position recognizer uses this FormattedData to give qualitative information (positional symbol) about the current position.
3. The energy recognizer uses this FormattedData to give qualitative information (energy symbol) about the current aircraft energy.
4. Quantitative and qualitative information from the FormattedData are gathered in an Event.

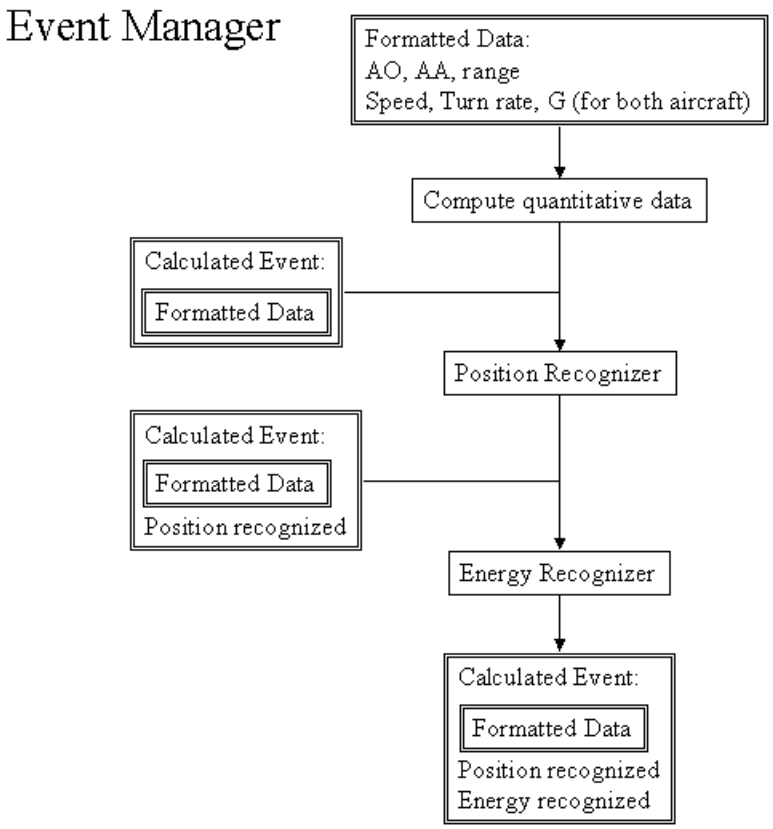


Figure 42: event manager principle

5.4 Skill-based level

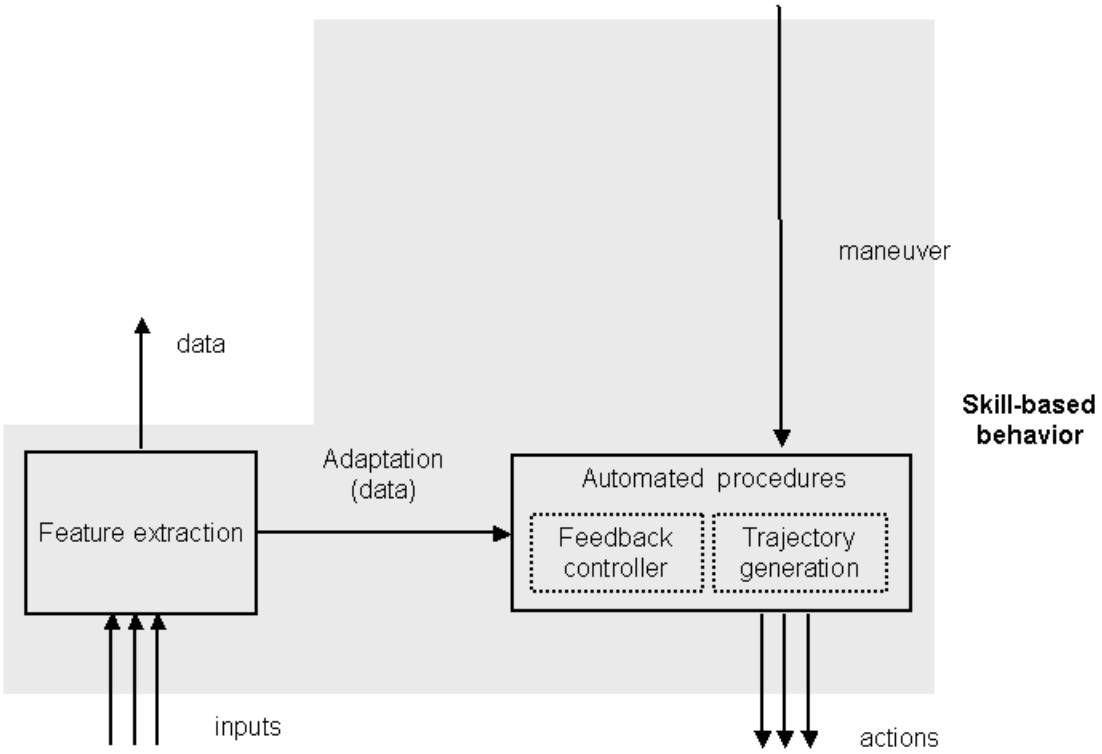


Figure 43: skill-based level in more details

Once a maneuver has been computed, the skill-based level should execute all the automated procedures that have to be performed. **Calculating trajectories** is one of the automated procedures that have been designed. The skill-based level computes all the corresponding inputs of the flight simulator. Next, it compares the output of the simulator to the expected values. This is called **feedback controlling**.

5.4.1 Trajectory generation

Once a maneuver has been chosen, a trajectory that corresponds to this maneuver has to be calculated, so that the bot can follow it in order to realize the maneuver.

5.4.1.1 Generating a trajectory:

1. A maneuver defines the *goals* of the bot and *policies* (term defined in section 4.2.2) tell how to reach them. For instance, a goal can be " turn rate = 15°/s", and the associated policy could be "Increase your turn rate".
2. For each policy and each corresponding goal, a mechanism is set that states the mathematical equations that will be used to calculate the parameter value that corresponds to the goal.
3. These mechanisms are used to compute all the trajectory points. By point it is meant the values of angle off, aspect angle, turn rate, speed and G the aircraft should have.

5.4.1.2 Choosing mechanisms:

A mechanism is associated to a policy. There are four kinds of policy, and consequently there are four kinds of mechanism. These kinds are:

- Angles
- Speed
- Turn rate
- G

For instance an angle mechanism is a set of functions to calculate angle off and aspect angle values. A simple one consists of a linear function that is set thanks to the current angle values (initial values) and to the angle goals (final values).

It was already said in section 5.2.2 that each policy has a priority. The policy with the highest priority begins to choose its mechanism. Then lower priority policy mechanisms can be influenced by higher priority policy ones.

5.4.1.3 Using the mechanisms:

To calculate one point, every mechanism has to be called, because each one calculates a different parameter value. Since the mechanisms that correspond to a high priority policy influence those that correspond to a low priority value, the calculation goes from the highest priority mechanism to the lowest one.

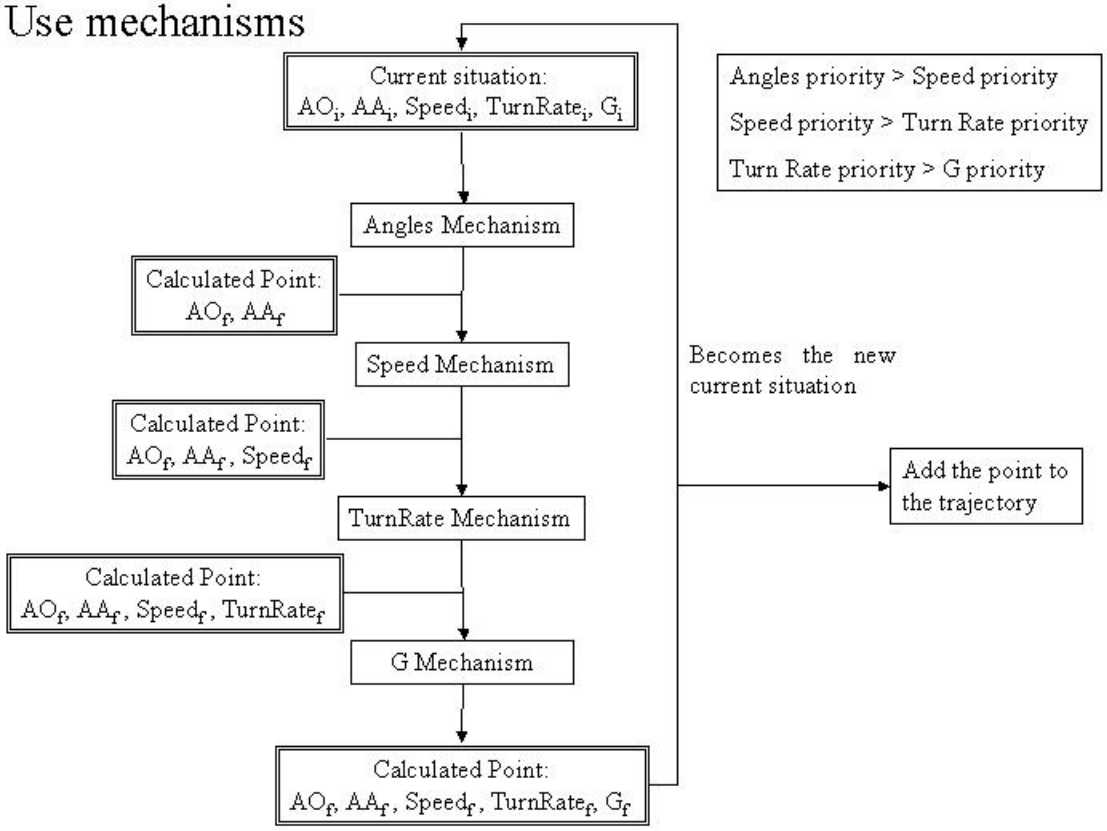


Figure 44: Using mechanisms for point calculation

5.4.2 Feedback controller

The Feedback controller purpose is to compare the expected point (from the trajectory calculator) to the real one (provided by the flight simulator) using a distance function.

1. Both points are compared parameter by parameter;
2. If for one parameter, the difference is big, then the feedback controller concludes that the real point is far from the expected one. Thus, the cognitive level has to take a decision.
3. Else if for one parameter, the difference is medium, then the feedback controller concludes that the real point is rather far from the expected one but it can be corrected. Mechanisms are re-initialized and the trajectory is recalculated;
4. Else, the feedback controller concludes that everything is ok.

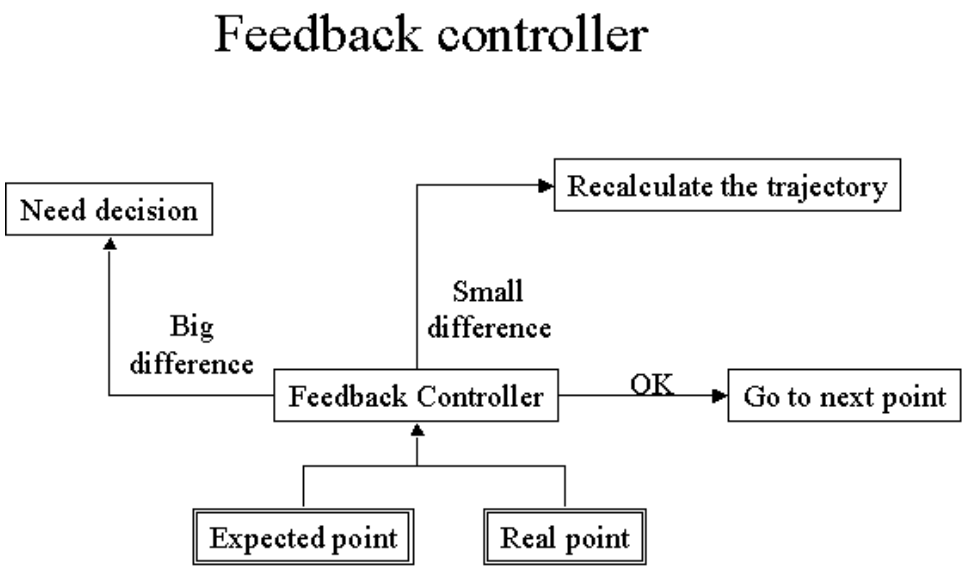


Figure 45: Feedback controller principle

5.5 Proof that it should work

What is expected is that the bot improves the positional configuration, that is to say angles and range. To show that the adopted approach works, there are two points of view. First showing that the outputs of the bot are computed in such a way that it must work. The other one consists of only showing that the cognitive outputs are computed in such a way that it works. Let us suppose that the skill-level does a good job. The second approach is less complete but it enables to validate the very level that we were in charge of, that is the cognitive level. The skill level implies expert knowledge we do not have.

Let us consider the cognitive level outputs. There are maneuvers:

```
MANEUVER
{
	DOUBLE _TARGETANGLEOFF;
	DOUBLE _TARGETASPECTANGLE;
	DOUBLE _TARGETSPEED;
	DOUBLE _TARGETTURNRATE;
	DOUBLE _TARGETG;
}
```

Considering the mathematical equations that links angles, range, turn rate, and G force (refer to paragraph 4.1.2) one can notice that they are not independent. To solve positional problems first, the bot has naturally to deal with angles and closure first. Thus, in the cognitive level, priorities have been set to be sure that main problems will be addressed first. From the highest to the lowest: angles, closure rate, turn Rate, G force. The priority system assures that the highest priority parameter will be processed the first to compute simulator inputs. This process will not depend on parameters of lower priority.

Besides the parameters of lower priority can depend on the ones of higher priority so that the associated goals might not be achieved.

So considering the cognitive model, maneuvers necessarily improves the current positional situation. Now, it is not as easy if we consider the outputs of the bot itself because they depend a lot on how simulator inputs are computed from maneuvers.

Chapter 6 Design and implementation

Now that all the algorithms and processes included in the bot have been precised, the current chapter will focus on the design and implementation of the program. First the design of the software in term of class architecture will be discussed the. Then some details about the coding and the testing of the software will be shown.

6.1 Layered architecture

The software design on the program is highly based on the bot design. Every level of intelligence corresponds to a software layer. The functionalities of the cognitive and skill level have been implemented in the AI layer and Know-How layer respectively. The Formatted data layer I an additional layer that is specific to the implementation of the bot because it is in charge of converting the data for the flight simulator.

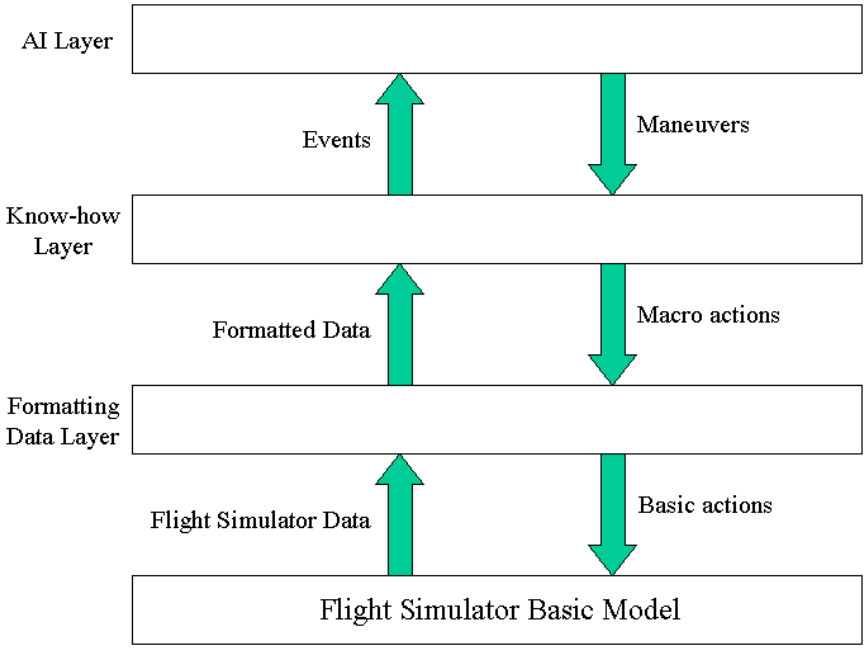


Figure 46: architecture diagram

The following chart maps the different terms that have been used and that represent the same ideas:

BOT DESIGN TERMINOLOGY	SOFTWARE DESIGN TERMINOLOGY
COGNITIVE LEVEL	AI LAYER
SKILL LEVEL	KNOW-HOW LAYER
POLICY	TACTIC (AI LAYER), KHPOLICY (KNOW HOW LAYER)
GOAL	PLAN
SET OF SYMBOLS AND ROUGH DATA	EVENT

6.1.1 AI layer

6.1.1.1 Functionalities

As input the AI layer takes the *event* computed by the know-how layer and provides the *maneuvers* as output. To achieve this task the following functionalities have been implemented:

- Finding out the candidate maneuvers by processing the *event* input.
- Dismissing the inappropriate maneuvers:
- Computing the final maneuver from the event inputs and the candidates.

6.1.1.2 Objects

Here are the main objects that have been created:

- **Maneuver**

A maneuver is the output of the cognitive level of the bot. The responsibilities are:

- Storing the goal values that should be reached at the end of the maneuver.
- Storing also the *policies* that have been chosen while the decision process is running. They should be ordered according to their priorities.

- **Maneuver Processor**

The responsibilities are:

- Calculating the maneuver features from the *event* input.
- Ordering the *policies* according to the priorities.

- **AI processor**

The responsibilities are:

- Launching the decision process.
- Launching the maneuver selection process.
- Launching the computation of the maneuver output.

- **Tree**

The responsibilities are:

- Coding the tree evaluation algorithm.
- Launching the object processes that need to operate during the decision tree evaluation.

- **Path**

The responsibilities are:

- Storing the path followed by the decision process: *Plan* and *tactics* are stored during the evaluation of a decision trees.
- Dismissing any other incompatible path.

- **Plan**

A *plan* represents a goal attached to a node of the decision trees. Its responsibilities are:

- Storing the goal encountered during the evaluation of a decision tree.
- Knowing the priority associated with the goal.

- **Strategy**

It is a set of *policies* attached to the leaves of the decision trees. The responsibilities are:

- Storing a set of policies.

- **Tactic**

Tactic represents a *policy*. After the decision tree evaluation, a goal is supposed to be associated to a set of *tactics*. The responsibilities are:

- Storing the *policy* that is attached to a goal. For instance "increase G" or "decrease speed". A policy is a constraint: the speed has to increase. Yet it is also an algorithm to compute the bot speed bot output for each point of the future trajectory. Nevertheless coding the calculation algorithm is not a responsibility of the tactic object in the AI layer. The calculation task belongs to the Know how layer.
- Knowing its priority.

The objects that have been described above correspond to the classes that have been created.

6.1.1.3 Activity diagram

The following diagram shows how these objects have been used to achieve the three main functionalities of the AI layer.

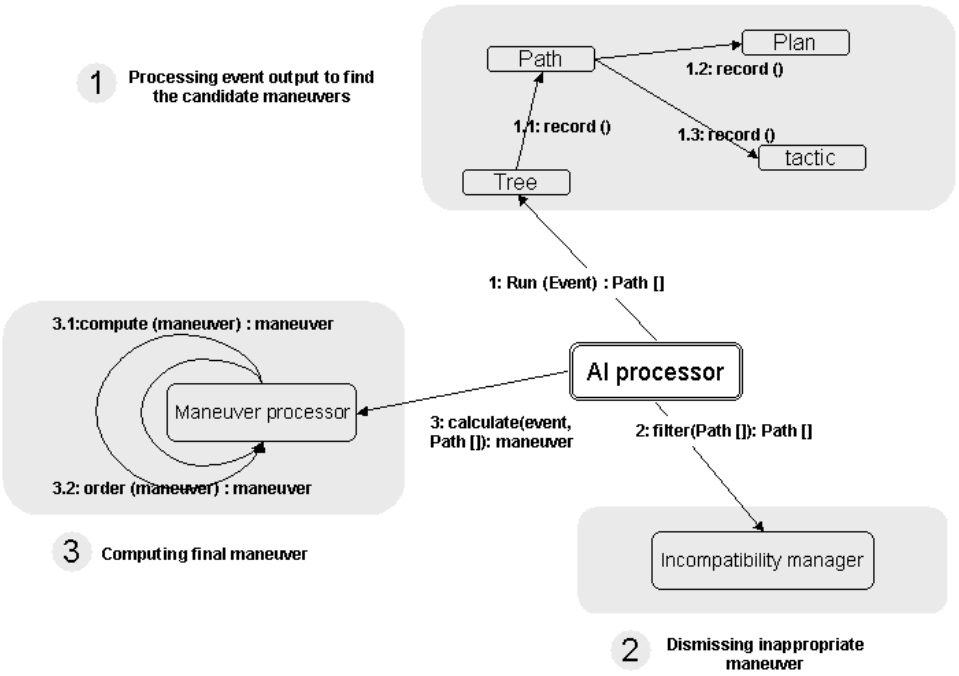


Figure 47: activity diagram of the AI layer

6.1.1.4 Class Diagram

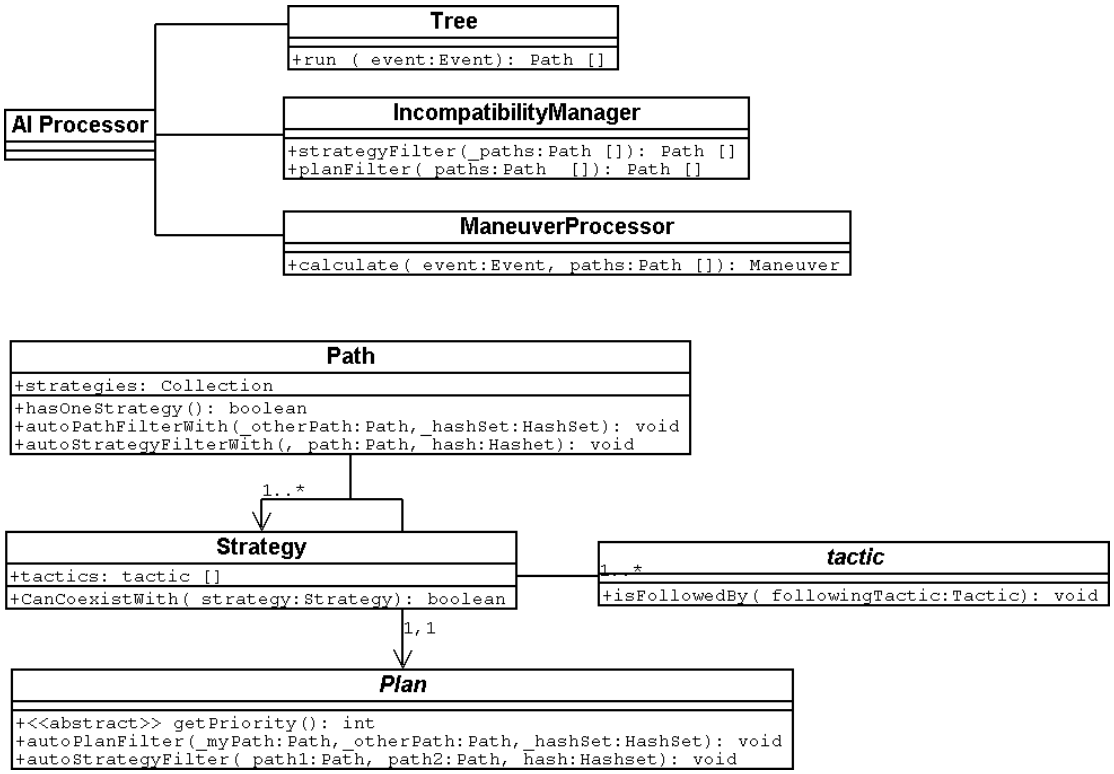


Figure48: AI Layer main class diagram

6.1.1.5 Sequence Diagram

The following sequence diagram depicts how the classes interact together when a maneuver is requested.

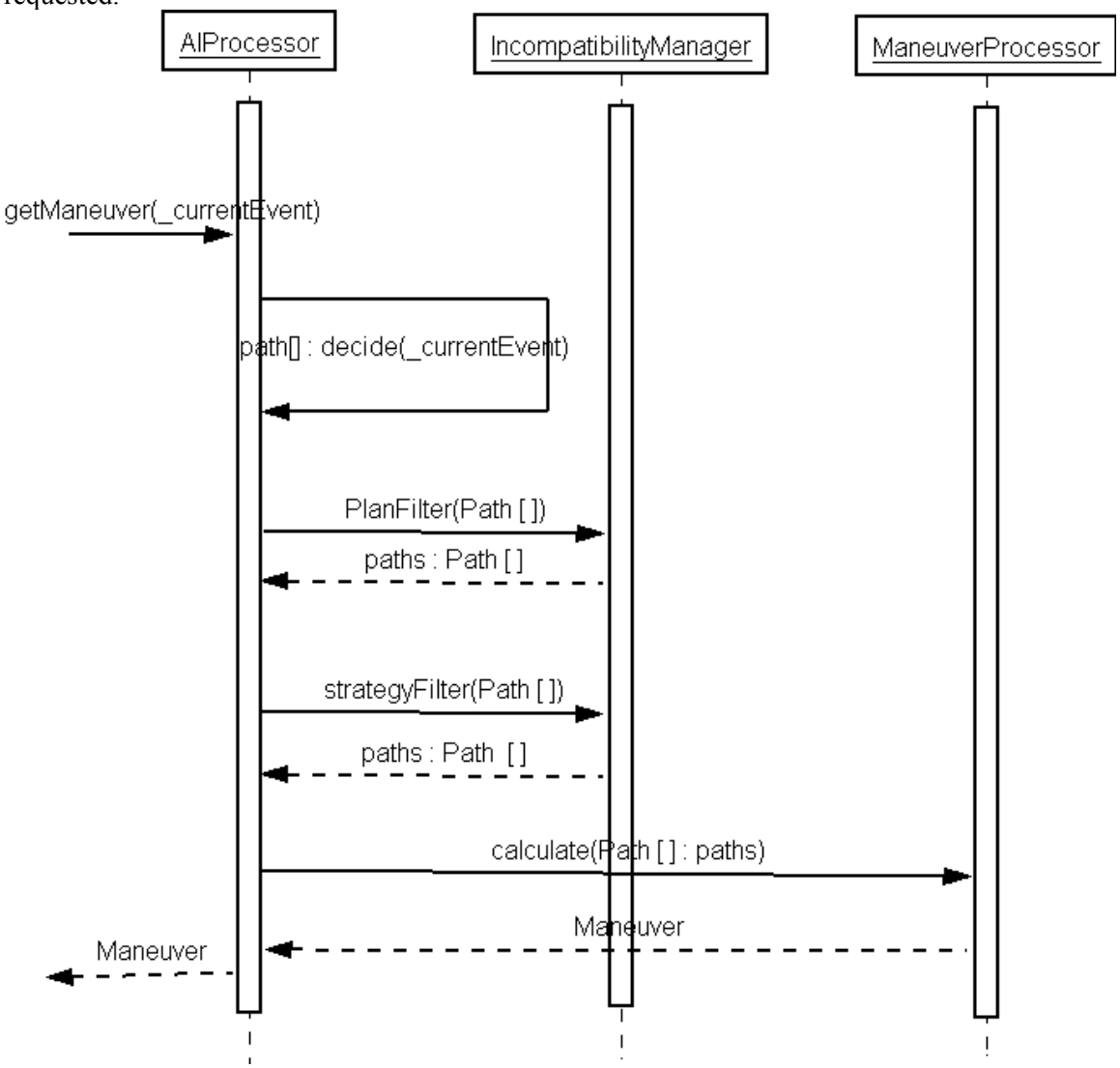


Figure 49: AI sequence diagram

The know-how layer calls the AI layer via the method *getManeuver()* with the current *event* as argument. Event is the bot representation of the environment.

- **Building the set of maneuver candidates**

The decision process starts by calling the method *decide ()*. An array of *paths* is obtained. They store the different branches that have been evaluated. The leaves of the branches represent the possible features of the future maneuver.

- **Building the most appropriate maneuver**

To dismiss the inappropriate features of maneuver two methods are successively called. *PlanFilter()* and *StrategyFilter()*. The first one removes from the set of maneuver candidates the ones the goals of which are incompatible. The second one only keeps the maneuvers the policies of which are compatible.

- **Computing the maneuver**

Finally the method *calculate*() is called with the compatible branches as arguments. A maneuver is returned.

6.1.1.6 Software Design details

- **Tree**

While the decision process is running, two objects operate to store information. *Path* stores the *plan* that represents a goal. It also stores the *tactics* that represent the candidate policies. To conclude while evaluating the trees, the potential features of the future maneuver are being computed.

- Saving the decision process branches in the path structure

Plan and *tactics* are attached to the nodes of the decision trees. Both implement the interface *Recordable*. It enables any objects to be recorded during the tree evaluation by using the method *record ()*. Thus, as soon as a node is entered any attached recordable objects are automatically recorded: *plan* and *tactics* are stored in the Path structure. The following diagrams depict these ideas:

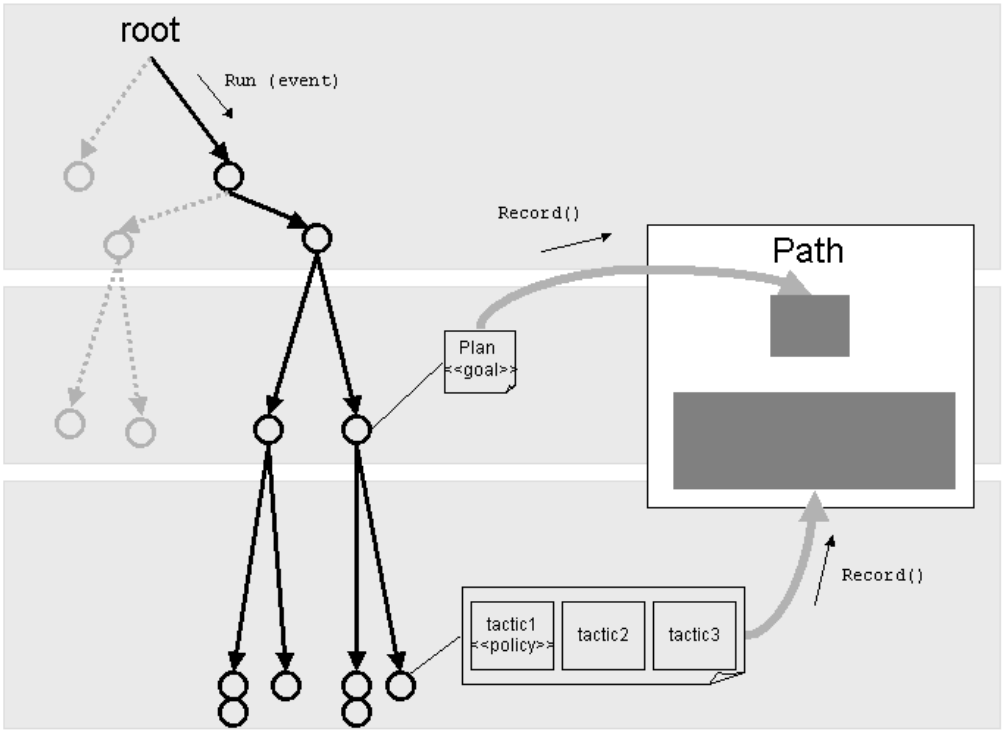


Figure 50: storing information during the decision process

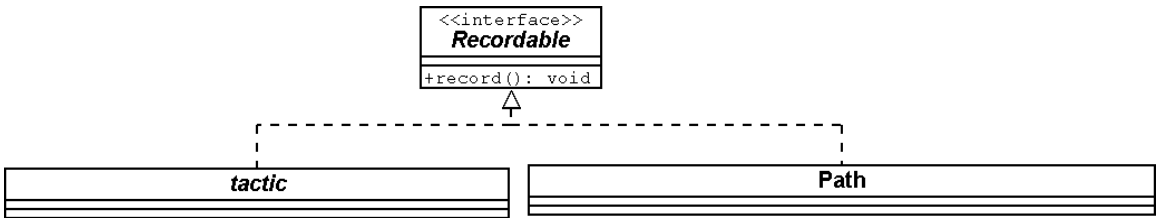
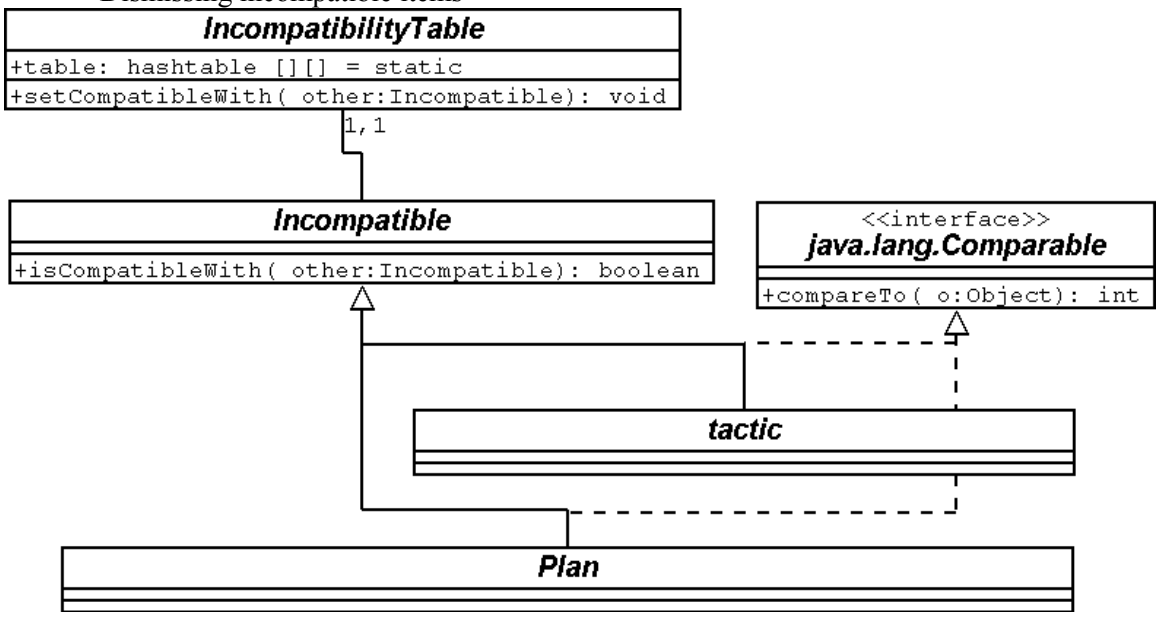


Figure 51: class diagram of the Recordable interface

- **Incompatibility manager**

To dismiss inappropriate maneuvers, the incompatibility manager operates before maneuver is computed. It dismisses the potential incompatible features that are about to compose the future maneuver. These potential features are passed as arguments to the filter methods. Next section deals with the ability to detect incompatible items.

- Dismissing incompatible items



The incompatibility manager needs to know if two *plans* or *tactics* are incompatible. "Being incompatible with" is not computed but this is knowledge. This information is stored in the bot memory. In fact, this knowledge is stored in a matrix. It maps two keys to a value. Keys are the objects that have to be compared that is to say *plan* and *tactic.* Values are booleans: *true* means that both objects are compatible *false* means that they are not. This matrix is implemented in the class *IncompatibilityTable.*

Any class whose instance has to be compared needs to implement the *Incompatible* class and the *java.lang.Comparable interface*. The *incompatible* abstract class provides the method *isCompatibleWith()* that enables any object to check if it is compatible with any other one.

During the initialization, every couple of compatible objects has to be set in the *incompatibleTable*. Any objects that are not in the table will be considered as incompatible.

It is very important to notice that objects are stored in the incompatibility table. Thus, two different instances of the same class can be incompatible if there are not set as compatible at the initialization.

- Dismissing incompatible paths

First of all, when two *paths* have incompatible *plans* they are declared to be incompatible. That is the role of the method *planFilter()*

Next, when two *paths* have incompatible *tactics* they are considered to be incompatible. That is the role of the method *strategyFilter ()*.

- **Maneuver Processor**

It processes *path* to compute *maneuver*. The input paths are necessarily compatible. It just has to compute the goal values and order the *policies* according to their priorities. This is the role of the method *calculate* ().

- Maneuver structure

This structure stores the output of the cognitive level. It includes the goal values:

```
DOUBLE TARGETANGLEOFF;
DOUBLE TARGETASPECTANGLE;
DOUBLE TARGETSPEED;
DOUBLE TARGETTURNRATE;
DOUBLE TARGETG;
```

The maneuver also stores the chosen policies:

```
TACTIC [] TACTICS;
```

- **Tactic**

The *tactic* class represents a policy. One should remind that a policy is associated with a goal and represents a possible way to achieve the goal. Thus a policy has a priority. A policy is also an algorithm used to compute parameters that have to be followed to achieve its corresponding goal. For instance, “increase the speed” is a policy. A possible algorithm could be “increase the speed of 20 knots every 1/10 seconds”. But the AI layer does not deal with any calculation algorithm, the know-how layer does. Consequently there are two different points of view for policy: the AI-layer view and the Know-how layer view.Given a policy, each layer should not have the same access to it.

Tactic represents the AI-layer point of view whereas KHPolicy represents the Know-how layer one. The following diagram illustrates this idea:

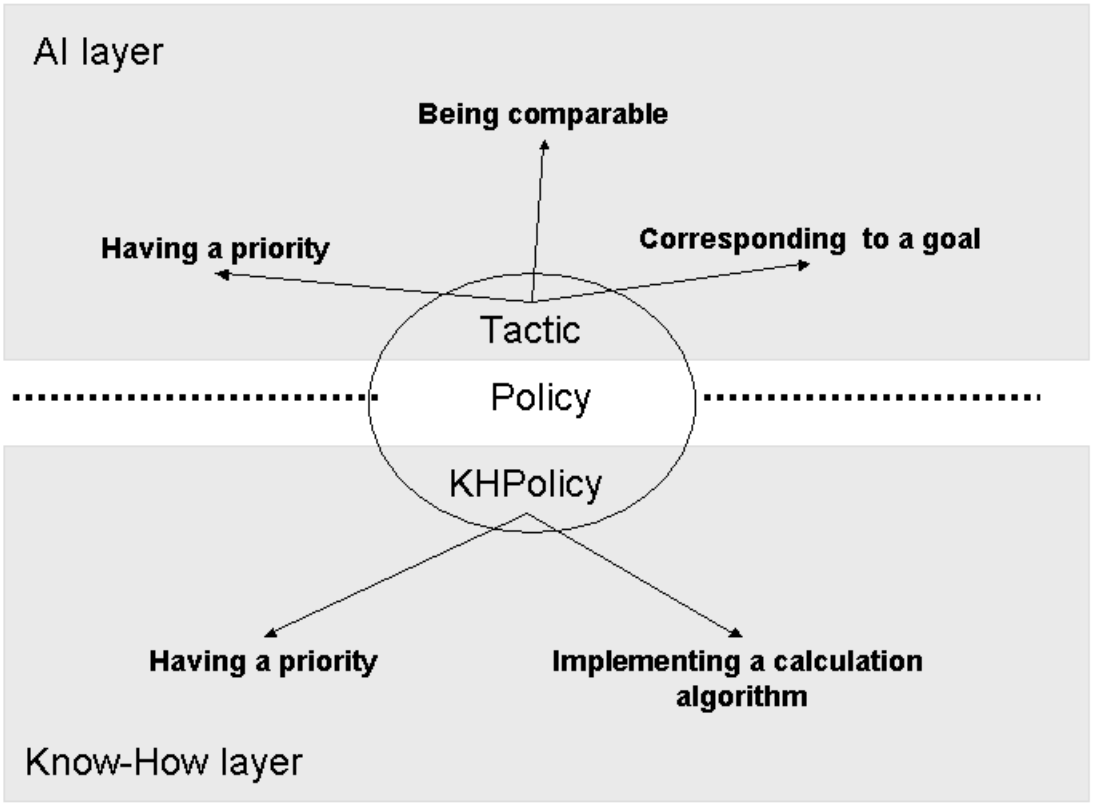


Figure 52: AI layer and Know-How layer views of policy

Currently there are four policy classes that extend the *Tactic* class: turnRatePolicy, Gpolicy, SpeedPolicy and anglePolicy.

Once a maneuver has been computed, goal values and policies have to be processed to provide Simulator engine inputs. It will be described in the next section.

6.1.2 Know How Layer

6.1.2.1 Functionalities

This layer contains all the bot's know-how skills. These skills are used in two different ways:

1. To compute action orders in a *MacroAction*, based on the *Maneuver* the AI has decided to execute.
2. To generate *Events* from the information provided by the **Formatting Data Layer** through a recognizing process. *Events* characterize what is happening.

6.1.2.2 Objects

Now are described the objects that have been created:

- **Trajectory Calculator**

It has the responsibility of:

- Calculating a trajectory, which corresponds to the *Maneuver* provided by the AI layer. It describes what is expected to happen but not what happens.
- Storing the points of the trajectory that has been calculated.

- **Policy**

It is linked to one of the aircraft parameters (i.e. angles (angle off and aspect angle), turn rate, speed, g).
It has the responsibility of:

- Defining the way the parameter it is linked to should evolve throughout the trajectory;
- Choosing and initializing a mechanism that will compute the values of the parameter throughout the trajectory.

- **Mechanism**

It is linked to one of the aircraft parameters (i.e. angles (angle off and aspect angle), turn rate, speed, g) and to a Policy.
It has the responsibility of:

- Calculating a value for the parameter it is linked to.

- **Point Provider**

It has the responsibility of:

- Providing the current trajectory next point to the lower layer.

- **Feedback Controller**

It has the responsibility of:

- Comparing what has happened (it receives data from the lower layers that have just updated the aircraft position thanks to the point provided by the *Point Provider*) with what was expected (i.e. the trajectory that has been calculated):
  - If it corresponds, then the next point is provided;
  - If there is a small difference, the end of the current trajectory is recalculated;
  - If the difference is too big, then a new decision is required and the *Event Manager* is called.

- **Event Manager**

It has the responsibility of:

- Calculating an *Event* that defines the current situation and providing the AI layer with it.

The following diagram shows how these objects work together::

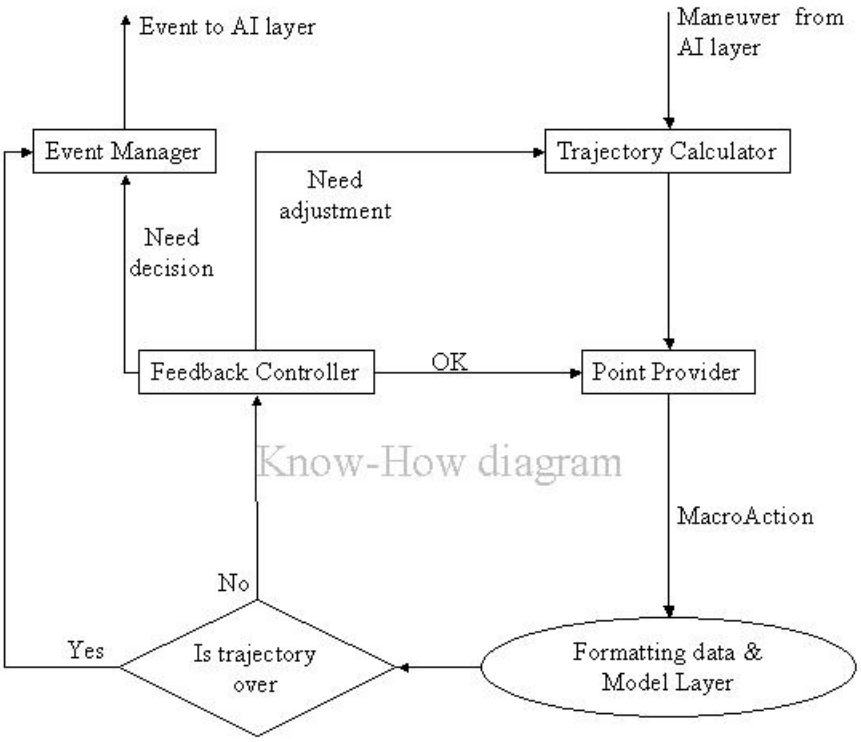


Figure 53: Know-How layer functionalities diagram

One can see that there are two kinds of objects:

1. The ones that interact with the AI Layer (*Trajectory Calculator* and *Event Manager*);
2. The ones that interact with the Formatting Data Layer (*Feedback Controller* and *Point Provider*).

Consequently the layer has been divided into two parts, two other objects have been created:

1. *AI Relay*: the part that interacts with the AI Layer;
2. *Model Relay*: the part that interacts with the Formatting Data Layer.

- **AI Relay**

It has the responsibility of:

- Managing the *Trajectory Calculator* and *Event Manager* objects. It links them both with the AI Layer and the *Model Relay*.
- All the communication between the Know-How Layer and the AI Layer:
  - It provides the *Trajectory Calculator* with the *Maneuver* computed by the AI Layer;
  - It provides the AI Layer with *Events* computed by the *Event Manager*.
- All the communication between *AI Relay* objects (i.e. *Event Manager* and *Trajectory Calculator*) and *Model Relay*:
  - It provides the next trajectory point, which is stored by the *Trajectory Calculator*, to the *Model Relay*;

- If the *Model Relay* asks for the trajectory to be recomputed, it orders the *Trajectory Calculator* to do so;
- If the *Model Relay* asks for a new decision, it orders the *Event Manager* to compute an *event* and provide it to the AI Layer.

❑ **Model Relay**

It has the responsibility of:

- Managing the *Feedback Controller* and *Point Provider* objects. It links them both with the Formatting Data Layer and the *AI Relay*.
- All the communication between the Know-How Layer and the Formatting Data Layer:
  - It gives to the Formatting Data Layer the next trajectory point provided by the *Point Provider*;
  - It gives to the *Feedback Controller* the current real point computed by the Formatting Data Layer.
- All the communication between *Model Relay* objects (i.e. *Feedback Controller* and *Point Provider*) and *AI Relay*:
  - According to the *Feedback Controller* result, it asks the *AI Relay* to provide the next trajectory point, to recalculate the trajectory or to ask for a new *Maneuver*.

The following diagram shows how the layer is structured:

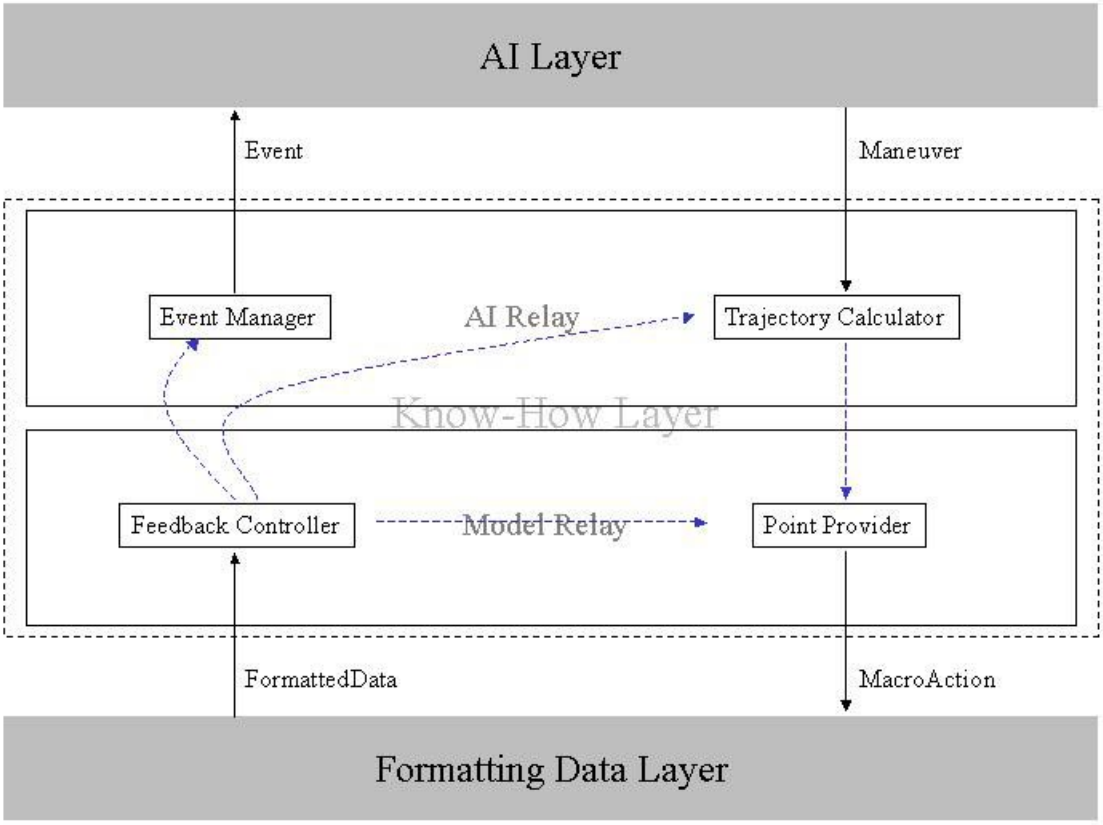


Figure 54: Know-How layer organization

6.1.2.3 Class architecture

The class architecture follows the layer organization and the objects definition. The AI Relay is mainly composed of three classes:

- *Trajectory*: used to calculate and to store a trajectory once mechanisms have been initialized;
- *EventManager*: used to calculate an event;
- *AIRelay*: manages the two other classes and links them with the AI Layer and the Model Relay.

Additional classes are needed to define all the possible mechanisms and policies.

The Model Relay consists in only one class, called *ModelRelay*, since its tasks are very simple. The Point Provider and the Feedback Controller have been gathered in the *ModelRelay*.

Now the main issues of each class are discussed.

- **FormattedData**

What for?: data coming from the flight simulator that have been formatted so that the bot can process them.

- **Event**

What for?: data that characterize the current situation and that are provided to the AI so that it can make a decision.

- **MacroAction**

What for?: actions order the bot has to perform to be able to reach the current trajectory point.

- **AIRelay**

What for?: the Know-How Layer part that communicates with the AI layer. Consequently it processes the maneuvers coming from the AI and provides the AI with events.

Data:

CurrentPredictedPoint: the Point that corresponds to the *MacroAction* that is currently processed by the Formatting Data layer

InitialFormattedData: the *FormattedData* that is used to (re)initialize the mechanisms before computing a trajectory.

CurrentManeuver

CurrentTrajectory

FirstPolicy: the policies are gathered in a linked list. This represents the first element.

Main methods:

CalculateTrajectory

It asks all the policies to choose and initialize its mechanism. Then it asks to the *Trajectory* to compute its points.

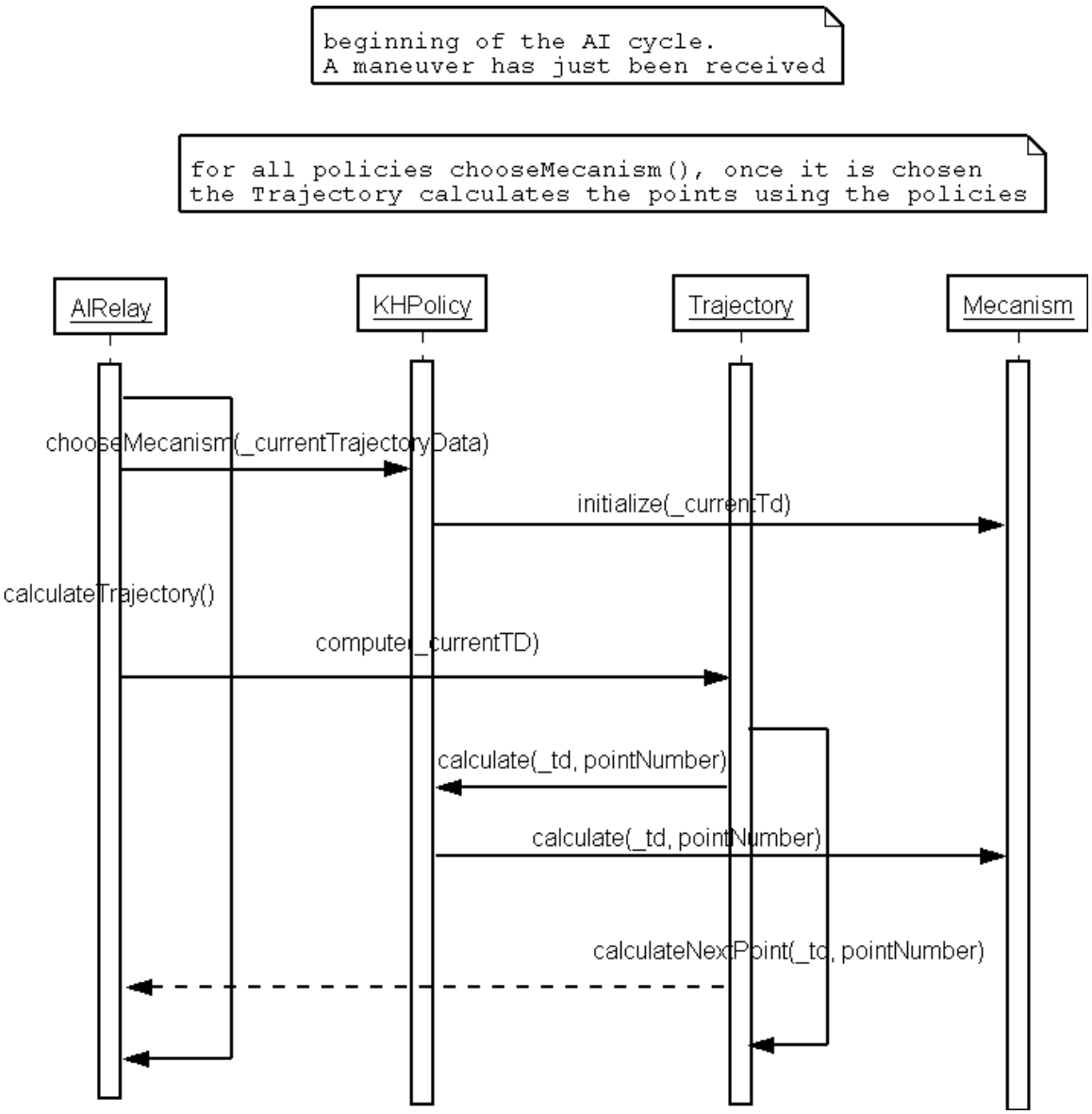


Figure 55: sequence diagram for calculateTrajectory

updateTrajectory
It does the same as **calculateTrajectory** except that the mechanisms are only re-initialized.

ProcessEvent
1. asks the *EventManager* to provide it with an *Event*;
2. provides this *Event* to the AI layer and gets the corresponding *Maneuver*;
3. sets the policies thanks to what is in the *Maneuver*;
4. calls **calculateTrajectory**.

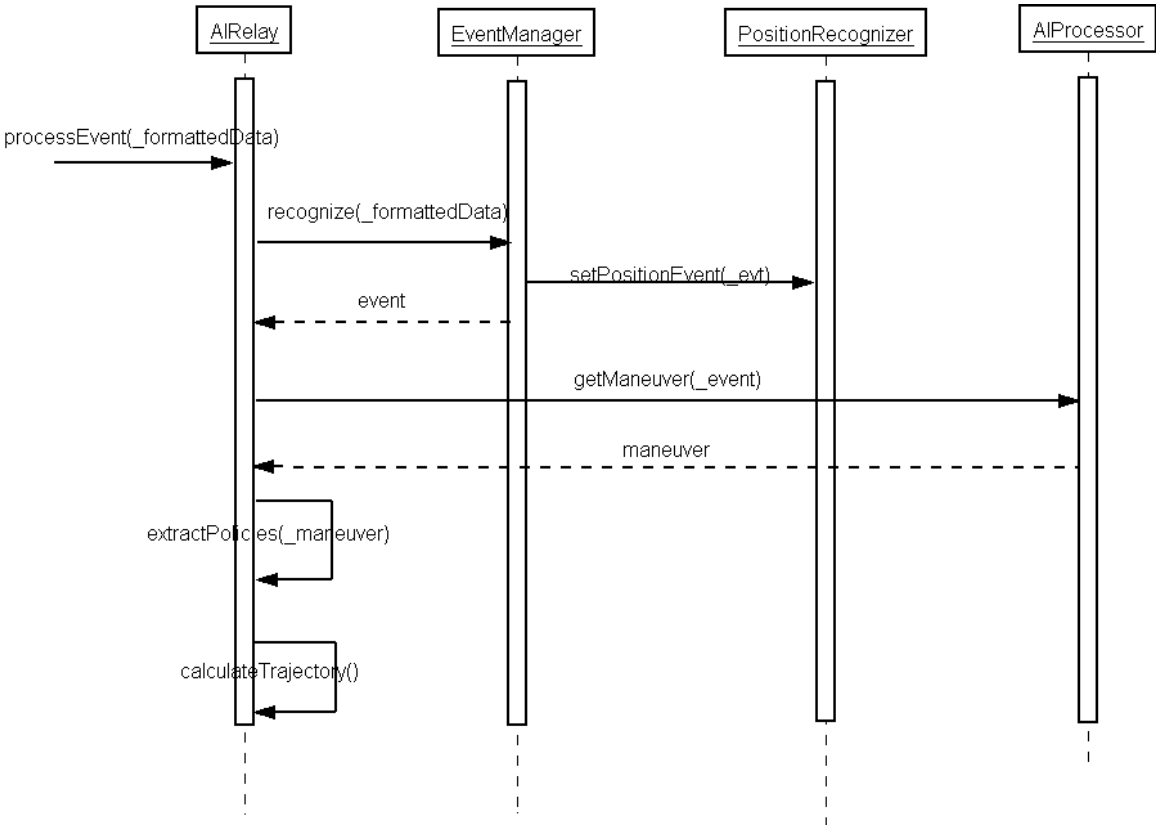


Figure 56: sequence diagram for processEvent

GetNextPredictedPoint
It returns the next trajectory point if there is one left and sets **currentPredictedPoint** to this value.

- **Trajectory**

What for?: used to calculate and to store a trajectory once mechanisms have been initialized
Data:

PredictionPeriod: the constant fixed time elapsed between two trajectory points.
PointsNumber: the trajectory point's number.
FirstPolicy: the policies are gathered in a linked list. This represents the first element.
Trajectory: the trajectory point's linked list.

Main methods:

Compute
Asks the policies to use their mechanisms to compute all the trajectory points (See Using mechanisms for point calculation);

- **Point**

What for?: data that define a trajectory point.

- **TrajectoryData**

What for?: data that are used during the calculating-trajectory process. They can be used in two different ways:

1. Initializing the mechanisms: the 'start' data correspond to the state of the aircraft at the trajectory beginning, the 'end' data correspond to the goals the bot has to reach i.e. the expected state of the aircraft at the trajectory end.
2. Computing the points: the 'start' data correspond to the state of the aircraft for the last calculated point, the 'end' data correspond to the state of the aircraft for the calculating point. (see Using mechanisms for point calculation: the former point is used to compute the following one).

- **Policy classes**

What for?: it defines how a the parameter it is linked to a should evolve throughout the trajectory. Basically these strategies are: increase, decrease or keep the same.
Common Data:

Mecanism: the mechanism associated to the policy
NextPolicy: the next policy in the linked list

Common methods:

ChooseMecanism
Chooses and initializes the mechanism for this policy as it has been explained in Choosing mechanisms:

Reset
Reinitializes the mechanism. It is used when the trajectory has to be recomputed.

GetNextPolicy & SetNextPolicy
The policies are gathered in a linked list. These two methods work with the next linked list element.

Calculate
Asks the mechanism to calculate a value.

(IsConstraintSatisfied: not used)

The Policies family: since there are many policies, a lot of policy classes have been defined. These policies are gathered by parameters, i.e. there are angles policies, speed policies, etc…
The following diagram illustrates this architecture:

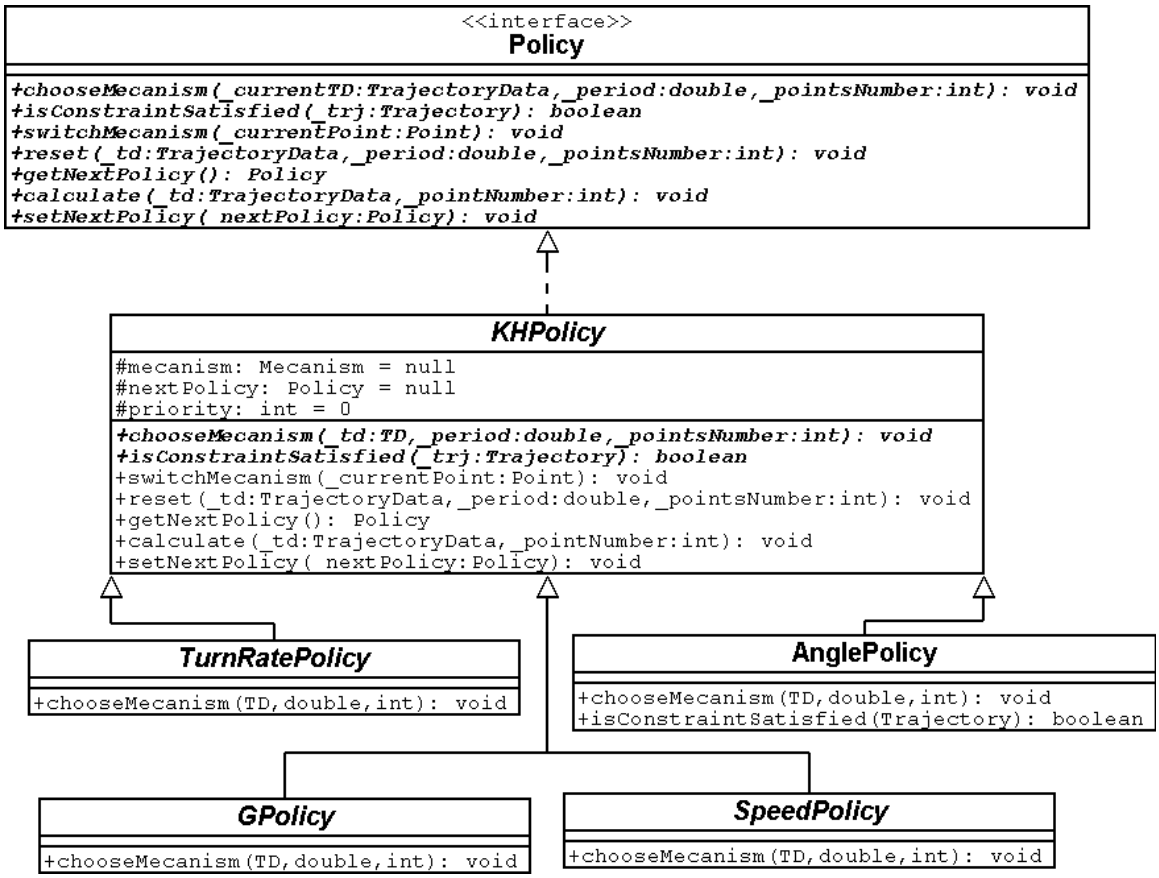


Figure57: policies main class diagram

The Policy interface: as it has been written, policy and tactic are two words that represent the same object but with different view. If 'tactic' is used the point of view is from the AI layer. If 'policy' is used, the point of view is from the Know-How layer. The *Policy* interface defines the Know-How layer point of view on these objects.
The *KHPolicy* class makes the link between the two viewpoints. It implements *Policy* and extends *Tactics*, which is the class that defines the AI layer viewpoint.
Consequently one will only find reference to *Policy* objects in the Know-How Layer.

- **Mechanism classes.**

What for?: equations used to set the point's parameter values when computing the trajectory.

Common methods:

Initialize

Given a TrajectoryData, it initializes some variables for the equations. For instance, for a linear equation, these variables are the initial value and the slope.

Calculate

Given a TrajectoryData, it calculates a value for the parameter it is designed to. This value will be used to create a trajectory point.

The Mechanisms family: since mechanisms are associated to policies, the same approach has been followed: the mechanisms are gathered by parameters, i.e. there are angles mechanisms, speed mechanisms, etc…

- **EventManager**

What for?: it processes a FormattedData, defining the current game world state, to recognize the situation.

Main methods:

Recognize

Given a FormattedData, it runs a recognizing position tree and builds the Event thanks to the running tree result.

- **ModelRelay**

What for?: the Know-How Layer part that communicates with the Formatting Data/Model layer. Consequently it processes the FormattedData coming from the FormattedData layer and provides it with MacroAction.

Data:

CurrentPredictedPoint: the Point that corresponds to the MacroAction that is currently processed by the Formatting Data layer

CurrentFormattedData: the last FormattedData that the Formatting Data layer provided. It characterizes the current game world state.

Airelay: a link to the AI Relay.

Main methods:

GetMacroAction

Called when receiving a FormattedData from the Formatting Data layer.

1. If the trajectory is over it asks the AI Relay to compute the event corresponding to the FormattedData. Consequently, a new *Trajectory* is defined.
2. Else it realizes the feedback control:
   - If a decision is needed, the AI Relay computes an event and sends it to the AI. Consequently, a new *Trajectory* is defined.
   - If adjustments are needed, the AI Relay re-initializes all the mechanisms and asks the *Trajectory* to be recomputed.
   - If everything is ok, the current trajectory is kept.
3. The next trajectory point is used to provide a MacroAction to the Formatting Data layer.

The following sequence diagrams show the four different executions for this method:

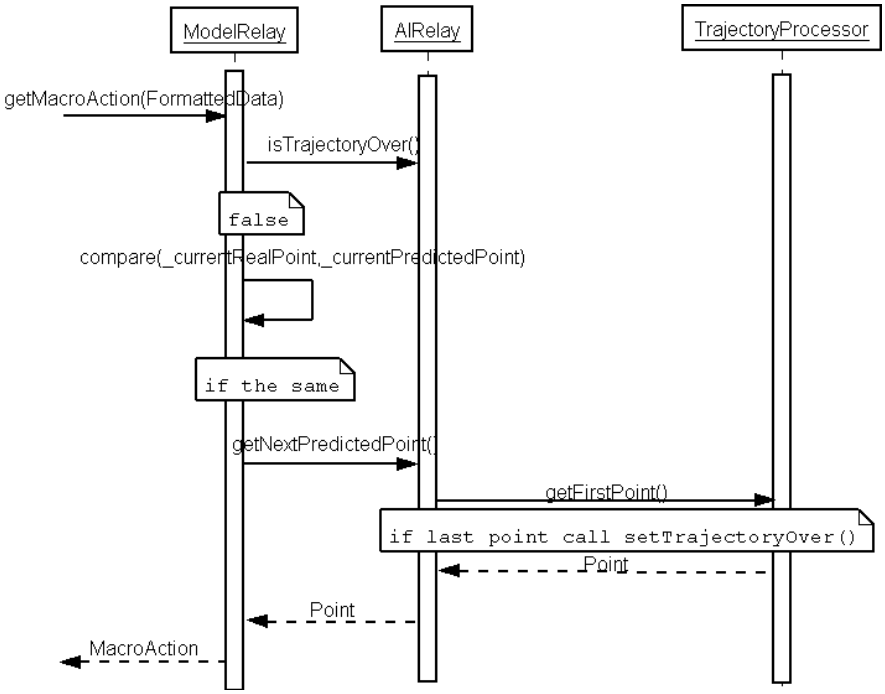


Figure58: case 'everything is OK'

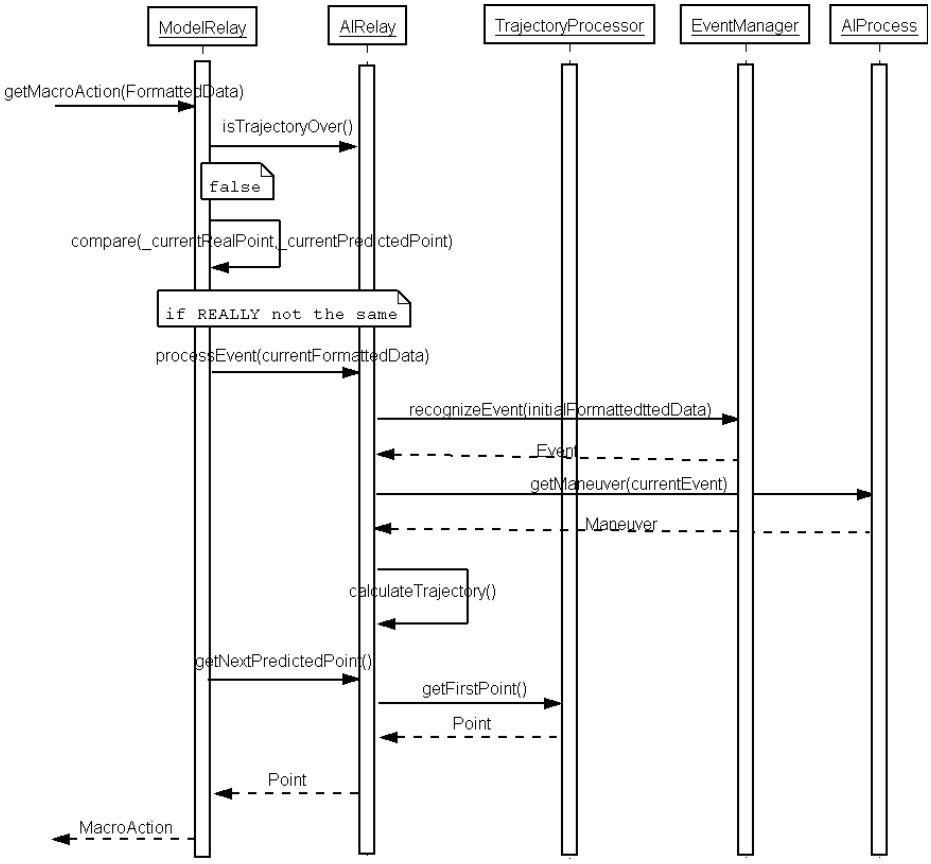


Figure59: case 'a decision is needed'

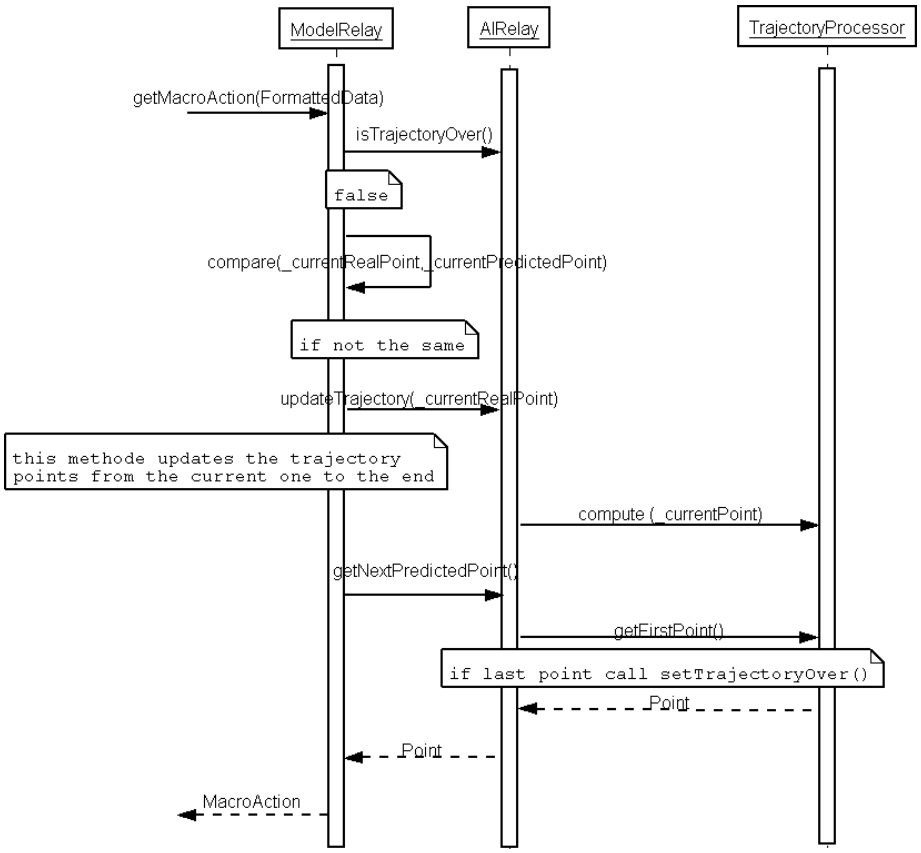


Figure60: case 'adjustments are needed'

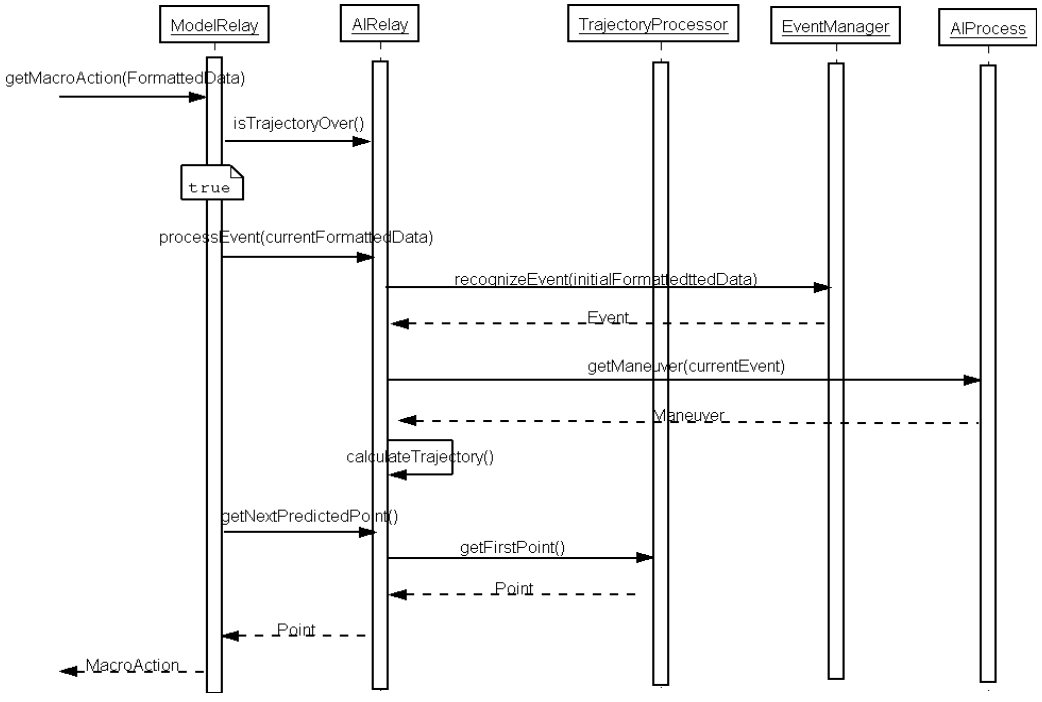


Figure61: case 'the trajectory is over

6.1.3 Formatting Data Layer

The formatting layer represents the link between the artificial player and the flight simulator. It is in charge of formatting the data coming from the flight simulator so that the bot can compute them, and formatting data coming from the bot so that the flight simulator can compute them. Consequently, this layer is highly flight-simulator-dependant.

The Formatting Data Layer (FD layer) has to perform two different tasks:

- Formatting the data from the know-how layer for the Model Layer, which represents the flight simulator.
- Formatting the data from the Model Layer for the Know-How Layer.

The following diagram shows how these two functionalities stand in the FD layer:

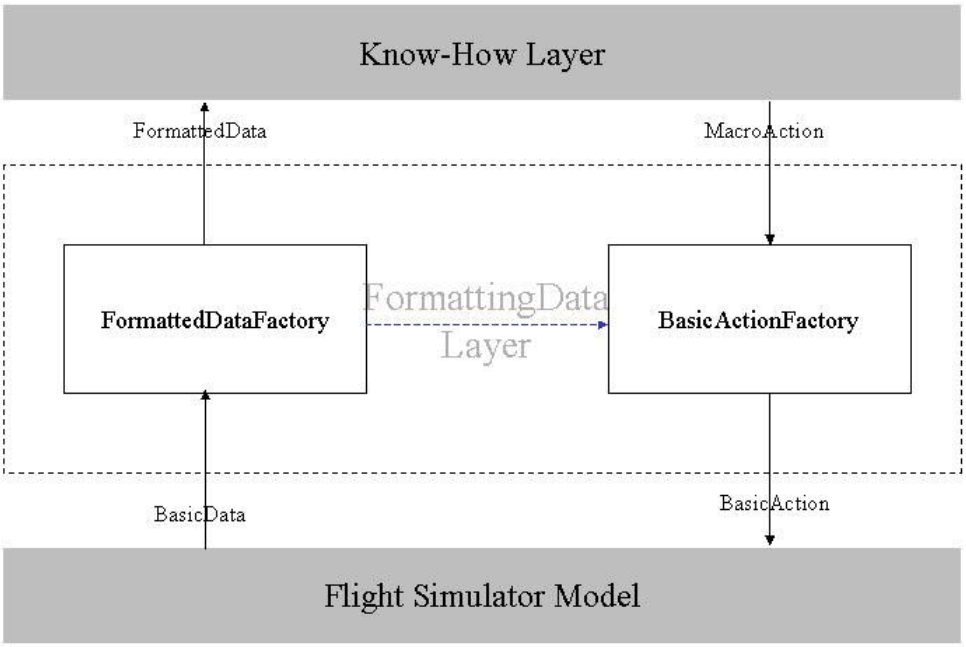


Figure 62: Formatting Data layer organization

Therefore this layer is composed of two classes, one per task.

- **BasicActionFactory**

What for?

This class translates the Know-How Layer output into input for the Model Layer. As shown in 6.1.2, the Know-How Layer outputs are *MacroAction* and as it will be shown in 6.1.4 the Model Layer inputs are *BasicAction*. Finally the BasicActionFactory is in charge of computing a *BasicAction* from a *MacroAction*.

How?

A *MacroAction* contains information about the next speed and turn rate the aircraft should have. A *BasicAction* (needed by the model to update the game data) contains information about the next aircraft turn Rate and its acceleration. So what the BasicActionFactory does is to compute the

acceleration needed to reach the speed that is specified in the *MacroAction*. Consequently it needs to know the current speed.

Input/Output

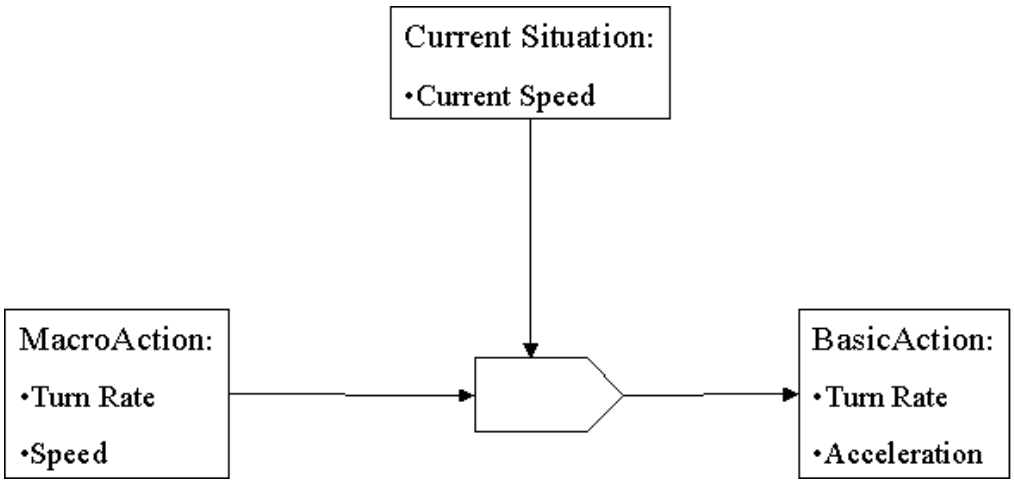


Figure 63: Basic Action Factory I/O

Information about the current situation is a *FormattedData* and is given by the *FormattedDataFactory* each time it computes a new *FormattedData* (i.e. each time the situation changes).

- **FormattedDataFactory**

What for?

This class translates the Model Layer output into input for the Know-How Layer. As shown in6.1.4, the Model Layer outputs are *BasicData* and as it has been shown in 6.1.2 the Know-How Layer inputs are *FormattedData*. Finally the *FormattedDataFactory* is in charge of computing a *FormattedData* from a *BasicData*.

Basically it consists in computing positional relative attributes based on positional absolute ones. These attributes are shown in the following diagram.

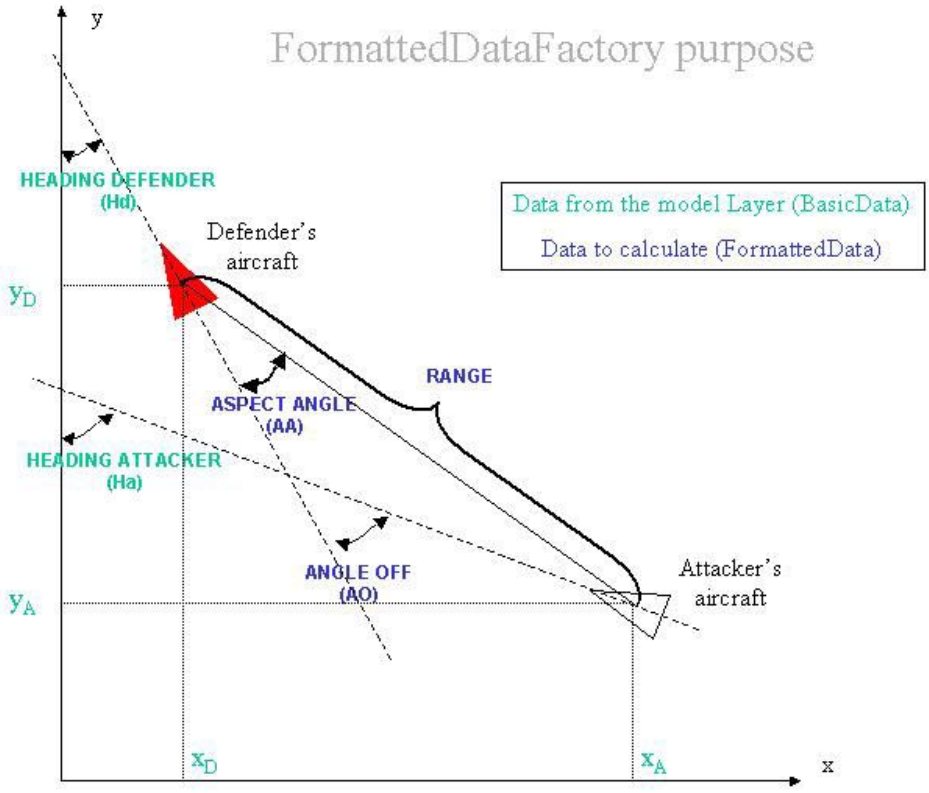


Figure 64: Formatted Data Factory purpose

How?

The calculation is based on geometrical considerations. To get the angle off and the range from the basic data is easy, whereas the Aspect angle calculation is a little bit more complicated:

- $AngleOff = Ha - Hd$
- $Range = \sqrt{(x_A - x_D)^2 + (y_A - y_D)^2}$
- $\cos(AspectAngle) = \dfrac{-(x_D + x_A)\sin Hd + (y_D + y_A)\cos Hd}{Range}$

Input/Output

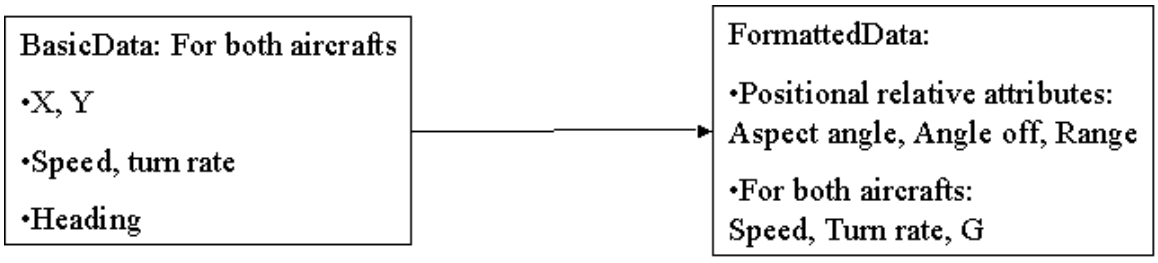


Figure 65: Formatted Data Factory I/O

6.1.4 Model Layer

Since it is time consuming to interface the bot with any flight simulator (very specific knowledge about how the flight simulator works is required) and since it is required to be able to check some results, a basic flight simulator model (FS model) has been developed. It is in charge of updating the game characteristics (aircrafts positions, speeds, turn rates…).
It receives some action orders from the Formatting Data layer via a *BasicAction*. Then based on these orders the new game characteristics are calculated.

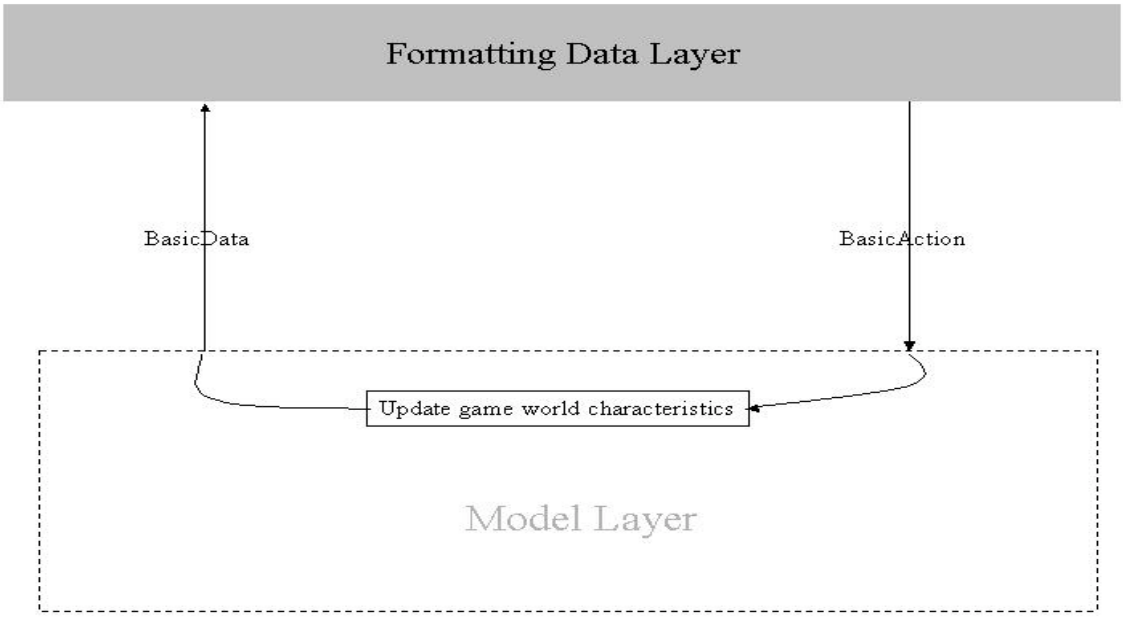


Figure 66: model layer organization

The Model layer class architecture is composed of three classes:

- Two data classes that represent the input and output of the model: *BasicData* and *BasicAction*
- The FS model itself

6.1.4.1 Data classes

- **BasicAction**

What for?
It represents the input of the FS model. All that the FS model needs to update the game characteristics is gathered in this class.

How?
Since the FS model is basic, it only needs a turn rate and an acceleration to update the game characteristics. That is exactly what one can find in a *BasicAction*.

- **BasicData**

What for?

It is the FS model output. It stands for the game characteristics.

How?

Since the FS model is basic, the game world has few characteristics. The game world is only defined by aircrafts positions, speeds, turn rates and headings.

6.1.4.2 The FS model class

- **FSBasicModel**

What for?

It has to update both attacker and defender aircraft based on a *BasicAction*. It is divided in two parts:

- Attacker's aircraft update: computing the new state based on the former one and to a *BasicAction*;
- Defender's aircraft update: computing the new state based on the former and to a predefined behavior.

Since the defender's behavior has not been modeled, the FS model cannot receive any order to update its aircraft state. Consequently a predefined behavior has been set that the FS model uses to update the defender's state.

How?

The following diagram shows what the FS model needs to proceed the calculation of the new characteristics:

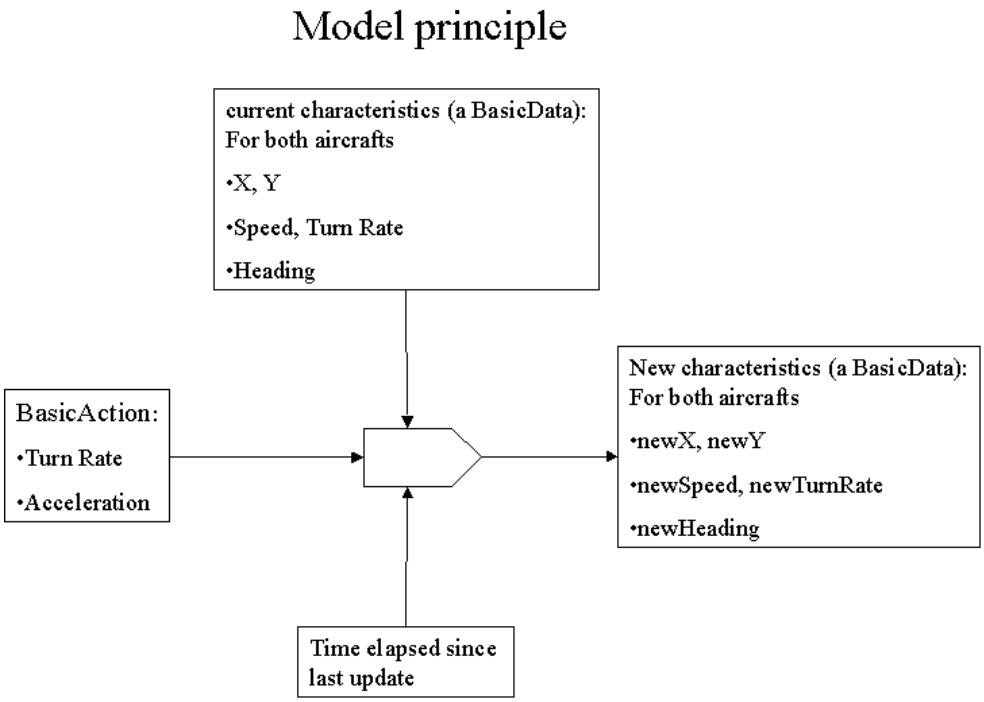


Figure 67: model principle

Three different kinds of data are needed to compute the new characteristics:

- The actions to perform (a *BasicAction* form the Formatting Data layer);
- The former characteristics (a *BasicData* it keeps in memory);
- The time elapsed since the last update (it is always the same).

For instance, for the new speed: $newSpeed = currentSpeed + Acceleration * elapsedTime$

Chapter 7 Future work

7.1 Taking energy into account

In the final release of the bot, it was planned to implement an energy recognizer. It should have mapped the current situation to energy symbols. They would have represented the pilot judgment about the energetic states of both aircraft. As this was planned from the beginning, it would be easy to add it.

7.2 A dynamic management of goals

In the current implementation, the goals were: addressing the turn Rate, angle configuration, range and G force problems. Each goal corresponds with a priority, which determines how it influences the maneuver computation. To make it more realistic, priorities should be dynamically changed during the run-time according to the current situation. For instance, if the bot is about to overtake the enemy the first priority problem is range. Everything should be done to address this crucial problem. This improvement should be quite easy given the software design.

7.3 Improving maneuver generation

To generate maneuver, a mechanism system has been implemented. A mechanism represents an algorithm used to translate a *goal* and *policy* to a set of actions almost directly understandable by the simulator engine. The implementation of such algorithms depends on expert skills. That is why the implemented ones are very basic. The more accurate they are, the closer to the reality the bot will be.

7.4 From recognizing to predicting

Now the decision process is based on the assessment of the current situation essentially. Due to the current energy and angle configuration, the most appropriate maneuver is computed to achieve goals. But a real pilot also takes into account the most probable situation to choose the next maneuver. With such ability, the reaction delay will be reduced and performances increased. Assessing the future situation means taking into account any unpredictable but probable failures (out of gas or gun damages), assessing the enemy's trajectory, etc. This last point is discussed further in the next section.

7.5 Influence of prediction on the decision making process

In the final release, the notion of prediction time has been introduced. It deals with how far the bot can foresee the future situation. It did not exist in the first draft of the bot.

To compute a maneuver, geometrical and energy parameters have been taken into account. Nevertheless, prediction about the enemy's trajectory along the prediction period should influence the decision process. It is not the case in the final release. Here follow some suggestions about how it could be taken into account.

In addition to geometrical and energy symbols, there could be symbols about the enemy's trajectory. They would represent the kind of maneuvers the enemy is likely to fly and how probable this maneuver is. The more probable the maneuver is, the more it should influence the bot decision. For instance, it could affect the prediction time. That is to say the more probable the enemy's maneuver is, the farther the bot should foresee. This is depicted by the following figures:

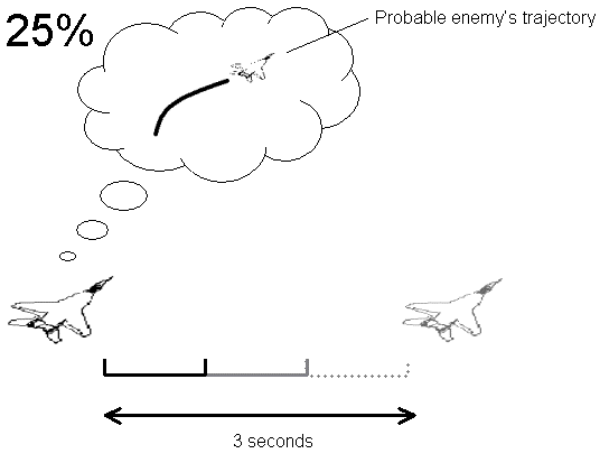


Figure 68: unreliable prediction

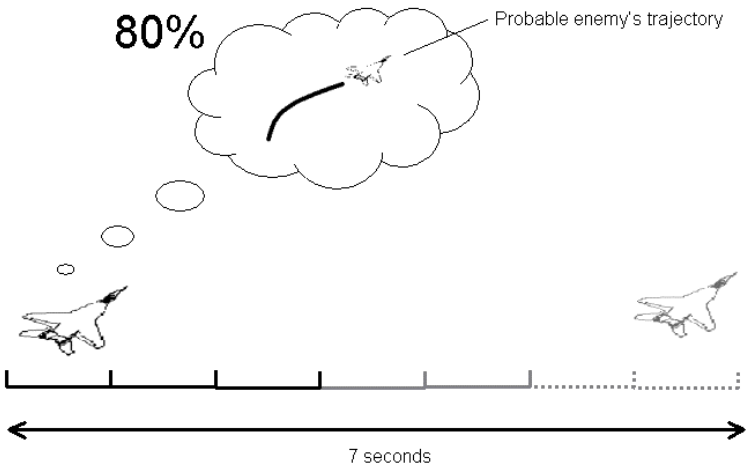


Figure 69: reliable prediction

Concluding remarks

The dogfight decision process was achieved as it was specified in the proposal. The decision making process was implemented. A rule-based approach was used. Decision trees were created to compute first *goals* and next *policies* that specified the way to reach them. To test and validate the decision making process, we needed to implement some additional abilities: recognizing the situation, calculating trajectories, feedback controlling, generating basic actions, processing data and simulating aircraft movements. Most of these abilities are very basic but work well. Nevertheless many difficulties arose.

To implement a complete flight simulator bot, it takes years. First, the most difficult task was to specify a model that could be implemented within the five-month project period. Next it was hard to acquire enough domain specific knowledge to make it feasible.

Firstly, we realized how difficult the modeling task was so that some requirements of the proposal were not considered that important. Even though the real time constraint is crucial for the artificial player, we were not able to check it. For the same reasons, we planned to make the program easy to personalize for someone that is not a computer science expert but focusing on this requirement turned out to be too time-consuming. Eventually we decided to concentrate on finding a solution that worked.

Secondly, the missing information issue was one of the main problems that we had to address. Interviewing domain experts such as pilots, cognitive science experts would have been useful, but it would have taken a lot of time to arrange these meetings. Consequently all data we got were from documents from the Internet such as F16 handbook, flight simulator specific websites etc. Nothing was complete, but enough information was available so that we could design a first pilot model.

Conclusive results are not available yet. Testing such a program is not an easy task. There are two points of view. Technically speaking, there was no obstacle. On the Artificial intelligence point of view, it is hard to certify that the used model is correct. First, it is impossible to implement a bot that functions precisely as a human being. That is to say that a bot behaves the same way as a human being to a certain extent. Such limits are crucial but quite difficult to set. Next, the validity of our work was hard to judge without any expert feedback. For sure, it would be interesting to focus on both of these issues in a future work.

Glossary

Lift vector	An aircraft lift vector is simply a vector that sticks directly out of the top of the jet, perpendicular to the aircraft's wings
Turn radius	It determines the size of the turn circle
Specific power (Ps)	A measure of an airplane's ability to gain or lose energy in terms of altitude, airspeed, or combination of both. Also called energy rate and expressed in feet per second or knots per second
Range.	It is the distance between two aircraft
Aspect angle	It is defined as the angle measured from the tail of the target to the position of the attacker.
Angle-off.	It is defined as the angular distance between the longitudinal axes of the attacker and the defender.
Turning room	It is the offset or distance from the bandit
Turn circle	It is simply the defender’s path that an attacker cuts through the sky when the defender turns.
Game plan	It is a sequence of maneuvers performed during a dogfight.
KCAS	Knots calibrated airspeed.
Prediction time	Time within when a maneuver should be entirely executed.

Bibliography

[Embrey] *Understanding human behavior and error,* David Embrey, Human Reliability Associates.

[Hand92] *MULTI-COMMAND HANDBOOK 11-F16, VOLUME 5, 10 MAY 1996,* Supersedes MCM 3-3 Volume 5, 1992

[Corad97] *Intelligent Agents for Aircraft Combat Simulation*, S. Coradeschi, L. Karlsson, A. Törne, Department of Computer and Information Science Linköping University, Sweden, 1997.

[Ehlert03] *The ICE project: The intelligent cockpit environment*, Ehlert, P.A.M., Rothkrantz, L.J.M, Delft University of Technology, 2003.

[EuroControl97] *Model of the Cognitive Aspects of Air Traffic Control*, Dr. K. W. KALLUS; Dr. M. BARBARINO, Ms D. VAN DAMME, European Organisation for The Safety of Air Navigation, 1997

[QuakeBot01] *The Quake III Arena Bot*, J.M.P. Van Waveren, University of Technology Delft, ITS, 2001

[SimHQ] http://www.simhq.com/

[AvionsLeg] http://avions.legendaires.free.fr/combat.php

[PsychoACM] http://www.ao.net/~chuck/atf/acmsycho.htm

Annexes

- *Specifications/Proposal*, Mahery ANDRIAMBOLOLONA, Pascal LEFEUVRE, University of Technology Delft, ITS, 2003.