Universiteit
van
Amsterdam

IAS technical report IAS-UVA-03-03

The Pursuit Domain Package

Jelle R. Kok and Nikos Vlassis
Informatics Institute
Faculty of Science
University of Amsterdam
The Netherlands

The pursuit (or Predator/Prey) domain is a widely used environment to test single or multiagent systems techniques. It consists of a grid world in which two types of agents, predators and prey, move around. It is the goal of the predators to capture the prey as quickly as possible.

This report describes our 'pursuit' software package, which can be used to implement and test the predator behavior in several scenarios without having to worry about other details involved in this domain.

We describe the following programs, which are all provided with the package: *pursuit* (main program), *pursuit_monitor* (visualization), *pursuit_logplayer* (playback tool) and the agent skeletons.

IAS
intelligent autonomous systems

Contents

Intelligent Autonomous Systems
Informatics Institute, Faculty of Science
University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam
The Netherlands
Tel (fax): +31 20 525 7461 (7490)
http://www.science.uva.nl/research/ias/

Corresponding author:
Jelle R. Kok
tel: +31 20 525 7524
jellekok@science.uva.nl
http://www.science.uva.nl/~jellekok/

1 Introduction

The pursuit (or Predator/Prey) domain was introduced in [Benda et al., 1986] and has become a widely used testbed for single and multiagent systems techniques. The pursuit domain consists of a discrete, grid world in which two types of agents move around: predators and preys. Figure 1 shows an example of a 10×10 world with two predators and two preys. In this world, it is the goal of the predators to capture all prey. In most cases, this domain has been studied with four predators and one prey, the latter moving randomly to one of its adjacent cells and standing still with a predefined probability such that the predators are able to catch up.

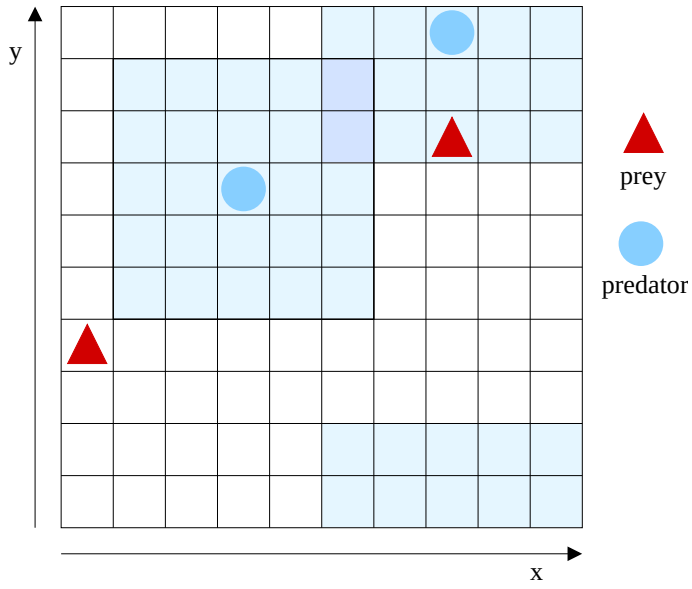

Figure 1: Graphical view of the world with two predators and two preys. The blue cells resemble the visible range of the predators.

Several variations of the original description have been studied over the years [Tan, 1993, Haynes and Sen, 1995, Korf, 1992]. We discuss the most important parameters related to these variations (parameters taken from [Stone and Veloso, 2000]).

Capture condition In its original setting, a prey is captured when all its adjacent cells are occupied by predators. Other possible capture criteria are surrounding the prey with two predators or occupying the same cell as a prey.

Visible range This denotes the number of cells from which a predator receives sensory information. Preys and predators that are outside this range are not visible.

Communication An important variation is whether the predators are allowed to communicate with each other and in this way be able to inform other predators of their strategies or sensory findings.

Legal moves Originally, an agent was only allowed to move to adjacent cells. A possible variation is to also include diagonal movements.

Grid-size The size of the world can be changed to different sizes. Furthermore, the world can be made planar (with borders on all edges) or toroidal. In the latter case the agents can directly move from one side of the grid to the other side.

Simultaneous or sequential movement This variation indicates whether the agents move at the same time or one after the other.

Prey movement In most variations the prey move randomly. Other variations would allow the prey to be more sophisticated and actively try to escape capturing.

For the practical of the course 'Multiagent Systems and Distributed AI'[1], the pursuit domain package has been used since spring 2001. In this package, we have parameterized many of the mentioned variations above, which makes it possible to easily change the characteristics of the domain.

The remainder of this report describes this package. We start in Section 2 with a short overview of the different components and descriptions of their usages. Thereafter, we describe the individual components in more detail. Section 3 describes the server details and the client protocols, Section 4 discusses the monitor that is used to display the world. Section 5 gives a description of the logplayer which allows for replay of recorded games. Finally, we end with a conclusion in Section 6.

2 Working with the Pursuit Domain

The pursuit package[2] consists of the following components:

Pursuit server This is the core program of the package. It models the complete environment and handles the connections with the client programs.

Monitor The monitor can be used to display the current world state of the pursuit server. After a monitor is started, it creates a connection with the server and then starts to receive information about the current world state. The monitor visualizes this information.

Agents The agents (both predators and prey) also create a connection with the server and then start to receive sensory information. Furthermore, they can sent actuator commands to the server, which causes the world to be updated accordingly.

Logplayer If the related option is turned on, the pursuit server logs all the subsequent world states to a file. The logplayer can be used to replay the contents of this file. A monitor is needed to visualize the current contents of the logplayer.

The remainder of this Section explains how to work with the mentioned components and how to create your own client programs.

2.1 Starting the environment

The pursuit server is the core program of the package; it models the complete world and handles communication with its connected clients. It can be started with a single command: `pursuit`. When started, it waits for client programs to make a connection. The pursuit server has no graphical interface, so to visualize the current state of the world, a monitor has to be started. This can be done with the `pursuit_monitor` command. The grid is then displayed without any predators or prey.

[1] More information about this course, can be found at `http://www.science.uva.nl/~vlassis/teaching/`

[2] This package is open source and can be downloaded from `http://www.science.uva.nl/~jellekok/software`. See accompanied files for installation details.

```
{Client structure}
continue = true
while continue == true do
  receive message from server
  if message == (quit) then
    continue = false
  else if message starts with (see then
    processVisualInformation()
    if determineCommunicationCommand() is not the empty string then
      send communication message
    end if
  else if message starts with (hear then
    processCommunicationInformation()
  else if message starts with (send_action then
    determineMovementCommand()
    send movement command to the server
  end if
end while
```

Algorithm 1: The client structure that shows the response to the different server messages.

2.2 Starting the agents

The agents (either predators or preys) can be started by connecting them to the server process. A skeleton implementation is available for both a predator and prey agent. It takes care of all communication with the server and defines several callbacks functions in which the behavior of the agent can be defined[3]. A predator is started with the command `predator` and a prey is started with the `prey` command. It is also possible to start multiple predators or prey; the only limit in the number of agents is the size of the grid.

After these programs are started, the corresponding agents appear on the monitor grid (predators are displayed as blue circles, prey as red triangles). Now a game can be started by pressing either the 'c' button or selecting the 'Continue' option from the mouse menu which appears when the mouse is clicked somewhere in the monitor. The predators and prey now start to move according to their defined behavior. The pursuit server automatically detects when a prey has been caught and will remove him from the grid. When all prey have been caught, the server gathers the statistics from this episode, places all prey and predators randomly on the grid and starts a new episode.

2.3 Implementing the agents

Each agent receives sensory information about its environment (relative position information to the other agents), can perform actions to move around in the world, and can communicate with other agents to coordinate its actions. The order in which this happens is fixed and is displayed in the basic structure of the client implementation in Algorithm 1. For more information about the protocol used between the server and the client agents, see Section 3.

Different callback methods are defined in this simple agent that have to be implemented in order to specify the behavior of the agents:

- `processVisualInformation`. This method is called with the received visual information as an argument. This string contains all relative position information of visible prey and

[3] The skeleton clients are available in C++, Java and Python but other programming languages can also be used as long as they support socket communication.

predators. By default this method only parses the incoming string and prints the position information of all visible agents to the screen. It can be used to create a model of the world as seen by this agent.

- `determineMovementCommand`. This method is used to specify the next movement command for this agent. The created command is then sent to the server. By default, this method returns a random movement command.

- `determineCommunicationCommand`. This method is only called when it is allowed to communicate with the other agents. In case the returned string is non-empty, it is sent to the server, which broadcasts it to all other agents when the communication option is turned on. The only restriction that applies to the communicated string is its length of maximum 256 characters.

- `processCommunicationInformation`. This method is called when a communication message arrives from another agent.

2.4 Starting the logplayer

When the corresponding option is turned on, the pursuit server stores the information about the current game into an output file. The logplayer can be started with the logged file as an argument, i.e., `logplayer game.log`. This starts a small window with some standard playback tools (play, rewind, forward, stop and jump to a specific cycle in an episode), which can be used to cycle through the recorded game. In order to visualize the contents of the logplayer, a separate graphical interface has to be started. This can be done by starting a monitor in exactly the same way as during a normal game.

2.5 Start script

A start script, `start.sh`, is provided that starts all relevant programs at once. By default, it starts the pursuit server, a monitor and four agents (two predators and two prey). It can be extended easily to start a different number of agents..

3 Pursuit Server

The pursuit server is the core of the package. It maintains the current world state and handles all communication with the agents. This section first describes how the server represents the world, how time is discretized in episodes and cycles, and finally discusses the agent protocols. The latter are used to handle the communication between the server and the clients. The server uses the protocol to provide the agents with information about their surroundings and the clients use it to inform the server of their intended action.

3.1 World

The world consists of a grid of $x \times y$ cells. Each cell has a distinct position denoted by its x- and y-coordinate. The cell in the lower left corner is labeled (0,0). The x-axis is the horizontal axis and the y-axis is the vertical axis. The upper right corner is thus labeled with the position (+x,+y). Agents can move around in the world to adjacent cells. The world is toroidal, meaning that the agents can move from one end of the grid directly to the other side of the grid.

The server processes the incoming actuator commands of the clients to update their positions on the field. The resulting state is then checked for the following special cases:

Prey caught When a prey is caught according to the specified capture condition, the prey is removed from the world. We have predefined the following four capture criteria:

- A predator and a prey share the same cell.
- A prey is surrounded by four predators. The predators are thus located north, south, east and west of the prey.
- A prey is surrounded by two predators. Thus two predators occupy two of the four surrounding cells of the prey.
- In the previous turn two predators were surrounding the prey, but only one moved to the location of the prey.

When a prey is caught, it is removed from the world.

Collision When two agents of the same type share the same cell, a collision occurs. In case of a collision both agents are penalized and placed at random positions on the field. In case of a collision between predators, both agents are colored differently for one cycle.

Penalty When a predator moves to the location of the prey while this is not allowed according to the capture criteria, the predator is penalized and placed on a random position on the field. Note that this case only holds for the fourth and last capture condition in which an agent moves to the location of the prey while in the previous turn he was not surrounding the prey together with another predator.

When all preys are captured an episode is ended and the number of cycles are recorded. All prey and predators are placed at random positions again and a new episode starts.

3.2 Episode and cycles

Time is divided into cycles. Each cycle consists of different stages in which either the prey or the predators are allowed to communicate with the server. Algorithm 2 displays all stages that the server uses. At the beginning of a cycle, visual messages are communicated to all prey after which the prey have `time_step` ms[4] to send a communication message to the server which is subsequently broadcasted to all other prey[5]. After this period has elapsed, the server communicates the `(send_action episode_nr cycle_nr)` message to the prey to indicate that the communication period is over and they now can send their movement command to the server. When again `time_step` ms have elapsed the sending period is over and the prey on the field are updated according to the received movement commands. Now it is the turn of the predators and the whole procedure repeats itself. The predators thus move simultaneously after the preys have moved simultaneously.

3.3 Agent Protocols

Each agent is controlled by a separate computer process. This client process creates a (socket) connection with the server. After the connection is created the server waits for an initialization message from the client. After this message is received, the agent starts to receive sensory information from the server and can send movement and communication messages that are executed by the server. All these messages have to obey a strict protocol, which is discussed next.

[4] When `time_step` is defined as -1 the server immediately continues when all agent messages are received. Use this value when a large amount of episodes have to be performed (i.e. for learning tasks).

[5] Note that when communication is turned off, this stage is skipped.

while not all preys are captured **do**
  {cycle}
  send visual information to all preys
  wait `time_step` ms to receive and process communication messages from prey
  send message `(send_action episode_nr cycle_nr)` to all prey
  wait `time_step` ms to receive movement commands prey
  update field
  check collisions
  send visual information to all predators
  wait `time_step` ms to receive communication messages from predators
  send message `(send_action episode_nr cycle_nr)` to all predators
  wait `time_step` ms to receive movement commands predators
  update field
  check collisions
  check whether a prey is captured
end while

Algorithm 2: Description of the different stages of the server during one cycle.

3.3.1 Initialization

After the socket connection is created, the agent can send an initialization message in the following format:

```
(init prey|predator)
```

The argument specifies whether this process represents a prey or predator agent. After a correct initialization message the server returns the message `(init ok)` and thereafter it starts sending sensory information and handle incoming messages. In case of an incorrect message, the server will not reply.

3.3.2 Sensory information

After initialization, an agent receives sensory information about the other objects in the world. This information has the following format

```
(see[ obj_info]*)
with
obj_info: (prey|predator x y)
```

The x and y value represent the relative distance in x and y direction respectively. In Figure 1 the predator located at position (3,6) receives the visual message `(see (predator 4 3) (prey 4 1) (prey -3 -3))` in case the world is fully observable. Objects are only located in the visual message when the relative distance to this object is smaller or equal than the distance defined in the `visible_distance` option (see Appendix A). With the default value of 2 for this parameter, the predator would observe no other agents and would therefore receive the message `(see)`. The value -1 corresponds to a fully observable world.

3.3.3 Movement commands

Agent can influence their environment by sending movement commands to the server. The server updates the positions of the agents by executing the received action. Movement commands have the following format

```
(move north | south | east | west |
      northeast | northwest | southeast | southwest |
      none )
```

The single argument specifies the direction in which the agent wants to move or `none` in case the agent wants to remain at his current location. In the default case, the diagonal movement commands are ignored and the agents can only move north, south, east, west or stand still.

3.3.4 Communication commands

If communication is turned on, agents can communicate with each other. A message should be sent to the server who broadcasts this message to all other agents. The communication message should be sent to the server in the following format:

`(say string)`

where string is a sequence of ASCII characters (maximum of 256 characters). The other agents immediately receive this message in the following format:

`(hear string)`

3.3.5 Referee commands

In special situations the 'referee' sends specific messages to the agents. This happens in the following situations:

- Episode has ended. The message `(referee episode_ended)` is sent to all agents.
- Two agents collide. The message `(referee collision)` is sent to the collided agents.
- A predator occupies the same cell as a prey, while the capture condition is violated. In this case, the message `(referee penalize)` is sent to the penalized predator.

4 Monitor

The monitor displays the current situation of the world and some statistics (episode and cycle number, average capture time and the total number of collisions and penalties). The monitor is a separate program that also makes a (socket) connection with the server. After each update, the server sends the information of all the agents and the statistics to the monitor. The monitor displays this graphically as is depicted in Figure 1.

4.1 Monitor Protocols

The monitor receives messages from the server that contain information about the world, but can also send commands that influence the server behavior. The syntax of these messages are explained next.

4.1.1 Initialization

After the socket connection is created, the monitor has to send the following initialization message:

`(init monitor)`

When this command is received by the server, it returns `(init ok)`. This message is immediately followed by the server parameter message.

4.1.2 Server Parameters

The server parameters which are also important for the monitor are communicated directly after the initialization confirmation. This message has the following format:

```
(server_param (visible_distance x) (rows r) (columns c))
```

where `x, r and c` represent the values from the corresponding server parameters.

4.1.3 World Information

The message that gives all information about the current world information has the following format:

```
(world (stats epis_nr cycle_nr avg_capture_time pen_nr coll_nr)
[((prey|predator nc) x y)]*)
```

The variables following `stats` resemble respectively the episode and cycle number, average capture time and the total number of penalizations and collisions. `n` is the number of this agent and `c` specifies the curent state of this agent (either `' '` in the default case, `c` in case of a collision and `x` in case of a captured prey).

4.1.4 User interaction

It is possible for the monitor to send commands to the server that influence the game. These commands are initiated by pressing specific buttons on the keyboard or selecting the item from the mouse menu which appears when the user clicks somewhere in the monitor. The different messages have the following format:

- `(monitor continue)`. Start/Resume playing. Select the 'Continue' option from the mouse menu or press the 'c' button.
- `(monitor pause)`. Pause the game. Select the 'Pause' option from the mouse menu or press the 'p' button.
- `(monitor quit)`. Quit the server. Select the 'Quit' option from the mouse menu or press the 'q' button.
- `(monitor step)`. Switch to step-by-step play. This means that the server advances one cycle and then waits for the next call to step-by-step. Select the 'Step-by-step' option from the mouse menu or press the 's' button.
- `(monitor speed_up)`. This option decreases the time waited before advancing to the next cycle with 10 ms (but not lower than 20 ms). This message is sent after the '+' button is pressed.
- `(monitor speed_down)`. This option increases the time waited before advancing to the next cycle with 10 ms. This message is sent after the '-' button is pressed.

5 Logplayer

The logplayer plays back logged games. When the server is started with the log option turned on, a human readable file is created that contains the received commands from the clients and the complete world information from each cycle. The logplayer parses this file and stores the relevant information internally. Furthermore, it provides an interface with the following control options:

play The logplayer cycles forward through all stored frames.

step The logplayer moves forward to the next frame and stops.

play backward The logplayer cycles backward through all stored frames.

step backward The logplayer moves backward to the previous frame and stops.

stop The logplayer stops playing and waits for another action..

jump The logplayer jumps to the specified episode, cycle number.

The logplayer sends the world information from the current displayed frame in the same format as the server to all connected monitors.

6 Conclusion

We have described the pursuit package which has been successfully used for the course 'Multiagent Systems and Distributed AI' at the University of Amsterdam since spring 2001. This software is an implementation of the pursuit (also called predator/prey) domain. Many aspects of the package are configurable, which makes it possible to test different variations of the problem.

References

[Benda et al., 1986] Benda, M., Jagannathan, V., and Dodhiawala, R. (1986). On optimal cooperation of knowledge sources - an experimental investigation. Technical Report BCS-G2010-280, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington.

[Haynes and Sen, 1995] Haynes, T. and Sen, S. (1995). Evolving behavioral strategies in predators and prey. In Sen, S., editor, *IJCAI-95 Workshop on Adaptation and Learning in Multiagent Systems*, pages 32–37, Montreal, Quebec, Canada. Morgan Kaufmann.

[Korf, 1992] Korf, R. (1992). A simple solution to pursuit games. In *Working Papers of the Eleventh International Workshop on DAI*, pages 195–213, Geneva, Switzerland.

[Stone and Veloso, 2000] Stone, P. and Veloso, M. (2000). Multi-Agent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robotics*, 8(3).

[Tan, 1993] Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the 10th International Conference on Machine Learning*, pages 426 –431.

A Configuration parameters

The server and the monitor can be configured using different parameters. Table 1 and Table 2 list all the configurable parameters for the server and the monitor respectively. The different configuration parameters can be passed to the server using a configuration file (for example `pursuit.conf` or `monitor.conf`). Provide the location of this file as an argument (`-conf conf_file`) when the program is started:

option	default	description
columns	15	number of columns
rows	15	number of rows
port	4001	connection port on server machine
time_step	100	time in ms for a phase in a cycle
nr_episodes	50	number of episodes before program halts
next_episode_wait	1000	time in ms waited after an episode
visible_distance	2	defines how far agents can see, -1 is fully observable
allow_comm	false	whether to allow communication
allow_diagonal_prey	false	is prey allowed to move diagonal
allow_diagonal_pred	false	is predator allowed to move diagonal
log_obj_info	true	indicates whether to log position info
log_obj_file	game.log	output file log info
automatic_start	false	start server automatically (delay 10s)
output_file	stdout	file to write episode times to
capture_method	1	method to capture prey (see earlier)
penalize_all	0	penalize all or only involved agents

Table 1: Server parameters

option	default	description
window_height	400	height of the window in pixels
window_width	400	width of the window in pixels
host	localhost	host on which server is running
port	4001	connection port on server machine
show_number	false	display agents' numbers
show_visible_range	true	display agents' visible range
background_color	ffffff	hexadecimal value of background color
grid_color	000000	hexadecimal value of grid color
prey_color	ff6633	hexadecimal value of prey color
predator_color	33ccff	hexadecimal value of predator color
caught_color	cc6633	hexadecimal value of caught prey color
collision_color	66cccc	hexadecimal value of collided pred. color

Table 2: Monitor parameters

IAS reports

This report is in the series of IAS technical reports. The series editor is Stephan ten Hagen (`stephanh@science.uva.nl`). Within this series the following titles appeared:

Joris Portegies Zwart, Ben Kröse, and Sjoerd Gelsema. *Aircraft Classification from Estimated Models of Radar Scattering.* Technical Report IAS-UVA-03-02, Informatics Institute, University of Amsterdam, The Netherlands, January 2003.

Joris Portegies Zwart, René van der Heiden, Sjoerd Gelsema, and Frans Groen. *Fast Translation Invariant Classification of HRR Range Profiles in a Zero Phase Representation.* Technical Report IAS-UVA-03-01, Informatics Institute, University of Amsterdam, The Netherlands, January 2003.

M.D. Zaharia, L. Dorst, and T.A. Bouma. *The interface specification and implementation internals of a program.* module for geometric algebra. Technical Report IAS-UVA-02-06, Informatics Institute, University of Amsterdam, The Netherlands, December 2002.

M.D. Zaharia. *Computer graphics from a geometric algebra perspective.* Technical Report IAS-UVA-02-05, Informatics Institute, University of Amsterdam, The Netherlands, August 2002.

J.R. Kok and N. Vlassis. *Mututal modeling of teammate behavior.* Technical Report IAS-UVA-02-04, Informatics Institute, University of Amsterdam, The Netherlands, August 2002.

J.J. Verbeek, N. Vlassis, and B. Kröse. *The Generative Self-Organizing Map: A Probabilistic Generalization of Kohonen's SOM.* Technical Report IAS-UVA-02-03, Informatics Institute, University of Amsterdam, The Netherlands, May 2002.

All IAS technical reports are available for download at the IAS website, `http://www.science.uva.nl/research/ias/publications/reports/`.