XPILOT

A differential game

and

A learning game Bot

Pablo Funes

INTRODUCTION

Xpilot

Xpilot is an internet client-server video game. The animation emulates a spaceship moving in (two-dimensional) space. Ships can rotate, use thrust to accelerate, and fire missiles. The purpose of the game is to score points by killing enemies without getting caught in their fire.

Xpilot was created by Bjørn Stabell, Ken Ronny Schouten and Bert Gijsbers in Norway. It is freeware. Contributions and additions are constantly being submitted by people around the world. The client and server programs together account today for more than 60000 lines of code, mostly uncommented.

Two computer game trends merge on Xpilot. On one hand, MUD's, MOOs, and the like, text games where players from different places in the internet interact with word commands and descriptions to create a virtual text world. On the other hand, space-ship and dogfighting games have been among the first computer games but had never been multi-player internet applications.

Xpilot consists of two computer programs: A client ("xpilots") and a server ("xpilot"). The server is where the game actually happens. This program runs the simulation and calculates the position and status of all the objects in the game at every time.

The client program is the one the individual players actually run to participate in the game. The client draws on a graphics screen a representation of the player's current position and orientation, along with nearby objects like other players, shots, and obstacles. A radar in a corner of the screen gives global information: The position of all the players and the map of the world.

Client and server communicate through the internet. During the game, the server program sends a full update every time frame, typically 15 times a second, to the client, which in turn interprets it in the graphic screen. The Xpilot client translates keystrokes into action requests that are sent to the server: Turn right or left, thrust, fire, etc. The server interprets them and the results will appear in future world updates. The client is passive, it does not attempt to calculate any effect of the player's actions, being limited to its sending keys, receiving status and rendering.

Xpilot community

Xpilot has generated a lot of interest in the internet community. There are several pages on the World Wide Web devoted to different aspects of the game, and the newsgroup rec.games.computer.xpilot where people can discuss their games, and suggest new variations to the current scenarios and new software features.

A problem with this game is the "Lag", as Xpilot users call the delays caused by slow network communications. Sometimes the connections of one or all the players to the server are not fast enough and the control becomes difficult. Tournaments between players in Europe and the US typically have this problem: Those who are in a different continent than the server will have a disadvantage.

This technological problem suggests that Xpilot is a bit ahead of its time. The current state of the art worldwide public network communications are not good enough yet to allow for this kind of real time dynamic virtual worlds to work fine.

I: A Differential Game

Isaacs [1] studied pursuer-evader games with ‘simple movement’, where pursuer and evader can choose their direction, and speed is constant.

Xpilot suggests looking at what we will call ‘second order simple movement’, where pursuer and evader can choose their acceleration direction instead of movement direction. Isaacs’ differential games are idealized models of real warfare situations, such as missile-anti-missile pursue. We propose a formal game that models pursue-evader situations between spaceships.

Constant speed arises in Isaacs’ games because of the presence of friction on earth: Cars, planes and missiles move at speeds that reflect an equilibrium of forces. A constant impulse force is equal in magnitude to an opposite friction force when a certain speed is reached. Even if Isaacs’ model of universal steering -where the direction of the velocity vector can be changed abruptly- is unreal, this model still seems reasonable to study some aspects of this kind of contest.

In the absence of friction, constant velocity ceases to have a practical sense. Constant force is still reasonable, as we imagine movement in space coming from reaction motors that will be able to exert a certain maximum force in the desired direction. As per Newton’s law, $F=ma$, a constant force implies constant acceleration.

We suggest modelling space pursuer-evader situations with *second order simple movement,* which results from considering either a constant or bounded acceleration as the control variable. Again, letting the direction of the impulse force to be changed at any rate is not realistic, but we think that this model will be useful to study some aspects of frictionless confrontations.

In Isaacs’ examples, the mathematical model gives a practical answer to the general PE game: The pursuer has to move in the direction of the evader, and the evader must move in the opposite direction of the pursuer. This is an optimal strategy regardless of the parameters of the game such as maximum pursuer and evader speeds, time or space limits. These parameters modify the value of the game but not the best strategy. Isaacs proves that these optimal strategies are effective even when the other player behaves non-optimally.

In Xpilot, all players have identical maximum accelerations. This means that the game value can be infinite: If P and E start from different positions $X_P(0)$, $X_E(0)$, and E accelerates at maximum rate away from P, there will be no capture. Still, we wish to proceed studying the mathematical game in the hope that the resulting strategies will have practical use in the actual game.

Our differential game is formally stated as follows:

STATE VARIABLES

X_P: The pursuer’s position
V_P: The pursuer’s velocity

X_E: The evader's position
V_E: The evader's velocity

GAME PARAMETERS

A: The pursuer's maximum acceleration
B: The evader's maximum acceleration

CONTROL VARIABLES

Φ: The pursuer's acceleration, $\|\Phi\| \leq A$
Ψ : The evader's acceleration, $\|\Psi\| \leq B$

EQUATIONS OF MOTION

$$\begin{aligned} X'_P &= V_P \\ V'_P &= \Phi \\ X'_E &= V_E \\ V'_E &= \Psi \end{aligned} \tag{1}$$

INITIAL CONDITIONS

$$X_P(0) = X_{P0}$$
$$V_P(0) = V_{P0}$$
$$X_E(0) = X_{E0}$$
$$V_E(0) = V_{E0}$$

PAYOFF

First t such as $X_P(t)=X_E(t)$ (time of capture)

The state and control variables, X, V, Φ and Ψ are vectors in two-dimensional space for Xpilot, or 3D space for a more realistic space game.

Let $\Phi(X_P, X_E, V_P, V_E)$, $\Psi(X_P, X_E, V_P, V_E)$ be strategies for both players. When both strategies are fixed, the game is determined by the system of differential equations defined by (1).

Once this system has been solved, functions X,V, Φ and Ψ are obtained that make true the following equations:

$$X_P(t) = X_P(0) + tV_P(0) + \int_0^t\int_0^u \Phi(s)\,ds\,du \qquad (2)$$

$$X_E(t) = X_E(0) + tV_E(0) + \int_0^t\int_0^u \Psi(s)\,ds\,du \qquad (3)$$

As the pursuer is by definition the minimizing player, the evader maximizes, and the game is finished when a capture occurs, the value of the game will be

$$\min_t \min_\Phi \max_\Psi \{ t \text{ such as } X_P(t) = X_E(t) \} \qquad (4)$$

With the maximum taken over an adequate set of functions φ, ψ (See [1], [2]). This setting does not give a clue on finding the solution of the game; for that we will use a geometric argument.

Considering the game from the point of view of one player only, suppose that the initial speed and velocity are zero (X(0)=V(0)=0). Given a fixed time t, the farthest point that can be reached will happen when the maximum acceleration α is applied in a constant direction. This means that the frontier of the reachable area for time t is a circle of radius $\alpha\frac{t^2}{2}$ as in the following picture:

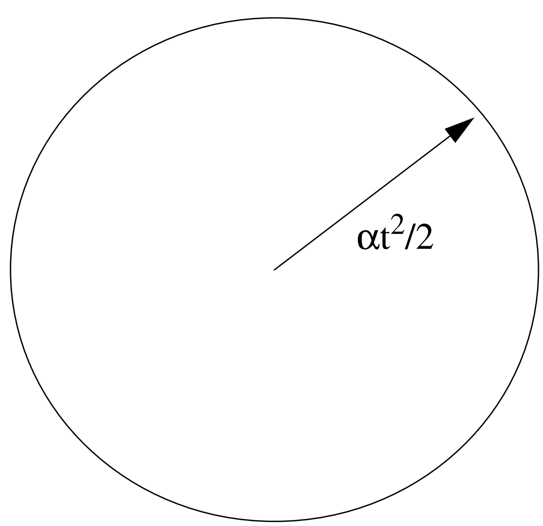


fig. 1

Depending on the chosen strategy, a player with initial position and velocity zero can be at any point inside a circle of radius $\alpha\frac{t^2}{2}$ at time t. The border of this circle represents constant maximum acceleration strategies: Those where the player has kept maximum acceleration at a constant direction throughout the time interval (0,t).

When initial positions X(0) and V(0) are considered, again a reachable circle is obtained, only shifted from the origin:

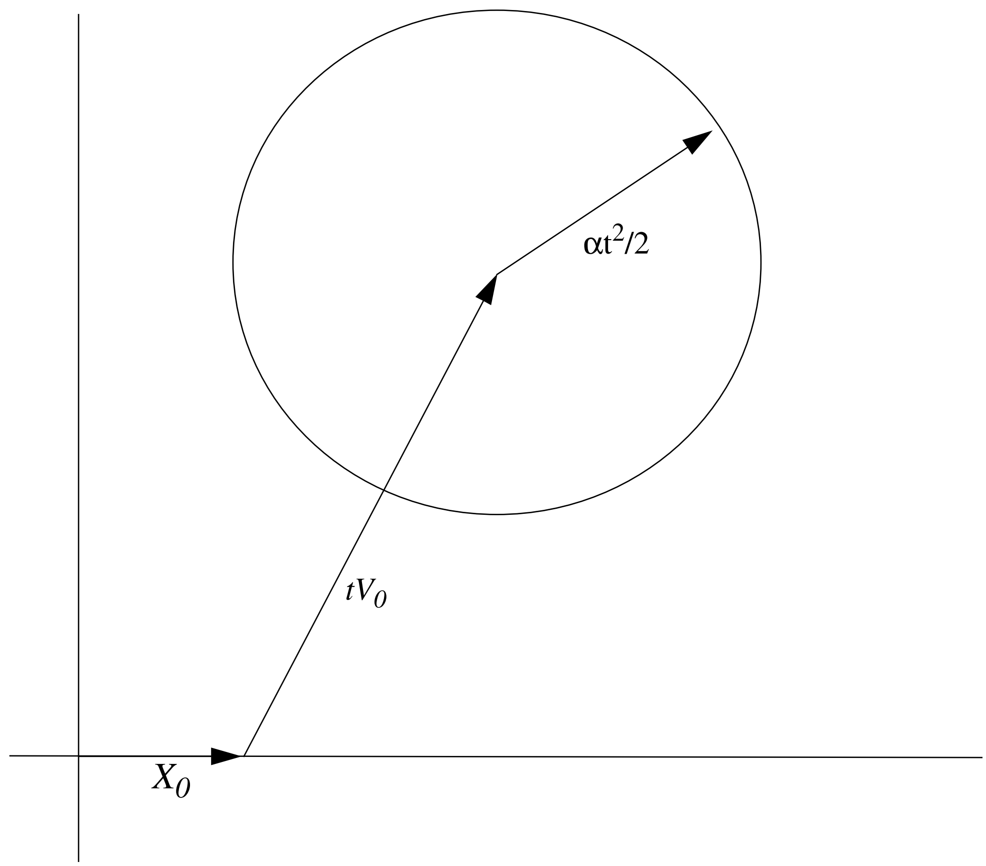


fig. 2

With this considerations we can proceed to solve the game. When pursuer and evader are considered simultaneously, the pursuer's initial speed and position $X_P(0)+tV_P(0)$ can be subtracted from all the remaining coordinates in order to simplify the picture. We can thus consider without loss of generality that $X_P(0)=V_P(0)=0$:

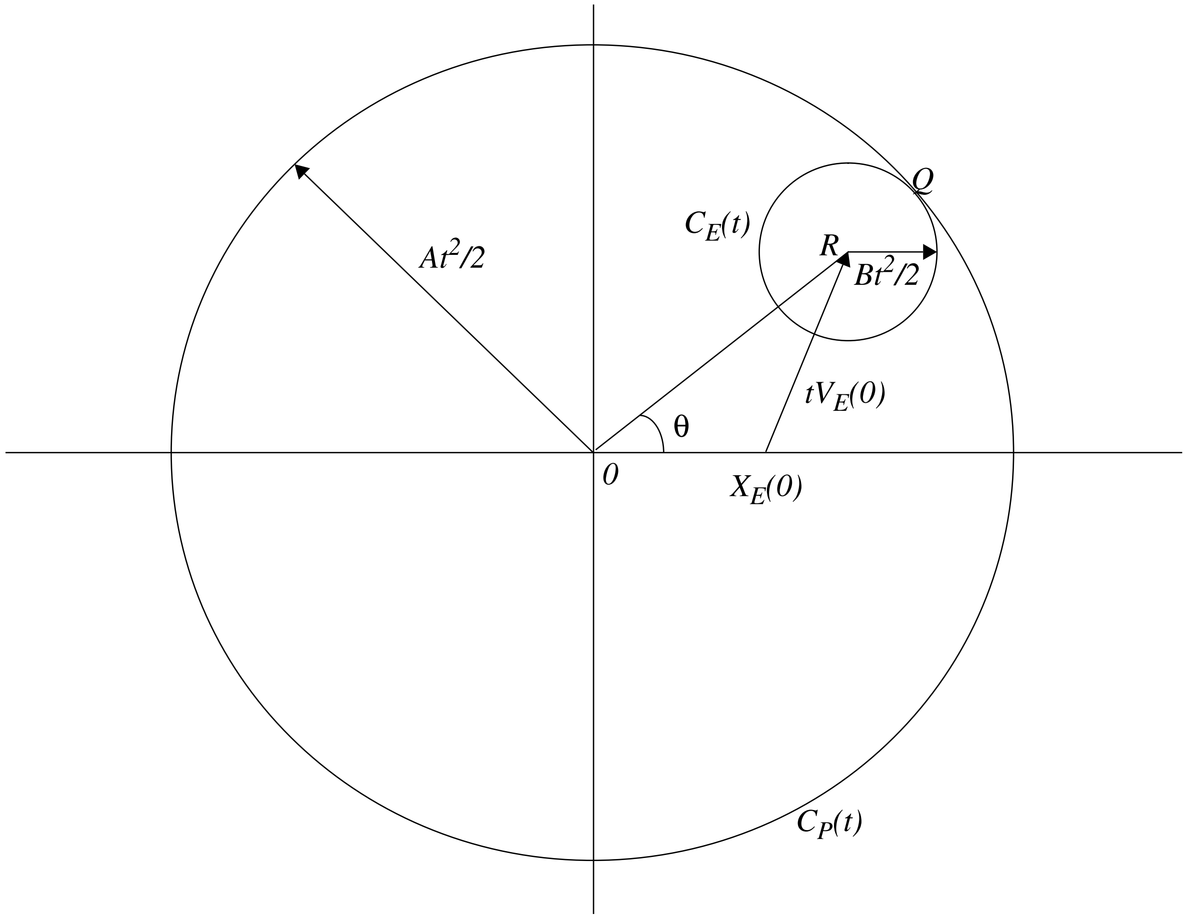


fig. 3

At time $t=0$ the reachable circles for the pursuer and the evader, $C_P(t)$ and $C_E(t)$ have radius zero. If the pursuer's maximum acceleration A is larger than the evader's B, then the circle for the pursuer will grow faster than the one for the evader, and eventually all of the smaller circle $C_E(t)$ will be inside the bigger $C_P(t)$.

As E is trying to maximize capture time, as long as not all of its reachable area $C_E(t)$ is inside $C_P(t)$, an optimal strategy for E will put him at time t inside its safe area $C_E(t)$-$C_P(t)$ to guarantee that capture will occur after t.

Let us consider the moment t when $C_E(t)$ is finally engulfed by $C_P(t)$. As the optimal strategy for E has been guiding him to the shrinking area $C_E(t)$-$C_P(t)$, at this final instant it will be at point Q. Considering this optimal strategy for E, its pursuer P will also head towards point Q.

The value of the game is consequently the time t that makes the circle $C_E(t)$ contained and tangent to $C_P(t)$. The optimal strategies for both players are constant in size and direction.

Let $V_E(0)=(a,b)$ and $X_E(0)=(c,d)$. With a rotation of our axis we can consider that d=0 as in fig. 3.

The center $C_E(t)$ is at $(c+at,bt)$ at any given time t. When t is the value of the game, circles $C_E(t)$ and $C_P(t)$ are tangent to each other, meaning that Q, $R=(c+at,bt)$ and 0 are aligned. A right triangle is formed by 0, $(c+at,0)$ and R whose hypotenuse has length $A\frac{t^2}{2} - a\frac{t^2}{2} = (A-a)\frac{t^2}{2}$. The value of the game is thus t such as

$$\left(\alpha t^2\right)^2 = (c+at)^2 + (bt)^2 \qquad (5)$$

Where $\alpha = \frac{A-a}{2}$.

Once t is obtained, the optimal strategy for both E and P is to accelerate at maximum rate with angle

$$\theta = \arctan\left(\frac{bt}{c+at}\right) \qquad (6)$$

As equation (5) is a fourth-degree polynomial, it is directly solvable. The solution obtained has, however, a long expression. Maple [3] finds t =

```
1/2

       2  1/6      2  1/6
   (2 a  3    + 2 b  3

      + 3

               1/3
         (1/27)

                 2  2      2  1/2       2  2      2  1/2    6  1/2
            (18 c  a  alpha  3    - 36 c  b  alpha  3    - a  3

                    4  2  1/2      2  4  1/2    6  1/2
               - 3 a  b  3    - 3 a  b  3    - b  3

               + 18

                  c

                         2      2  4    8      2      2  4       4      4
                     (- c  alpha  a  + b  + 8 c  alpha  b  + 16 c  alpha

                              2      2  2  2    6  2      4  4      2  6
                        - 20 c  alpha  a  b  + a  b  + 3 a  b  + 3 a  b)

                        ^ (1/2)

                     alpha)
```

```
                ^ (1/3)

        + 3...
```

(The printout of the whole solution takes sixty pages!)

Conclusions

The solution obtained for this problem does not have the niceness of Isaac's solution to the problem of simple movement. Here, the optimal strategy is dependent on the difference between the maximum acceleration rates of both players.

This model cannot thus be applied to Xpilot as is, because α is zero and in this case there is no optimal strategy suggested. A suboptimal model could be essayed as per the following figure:

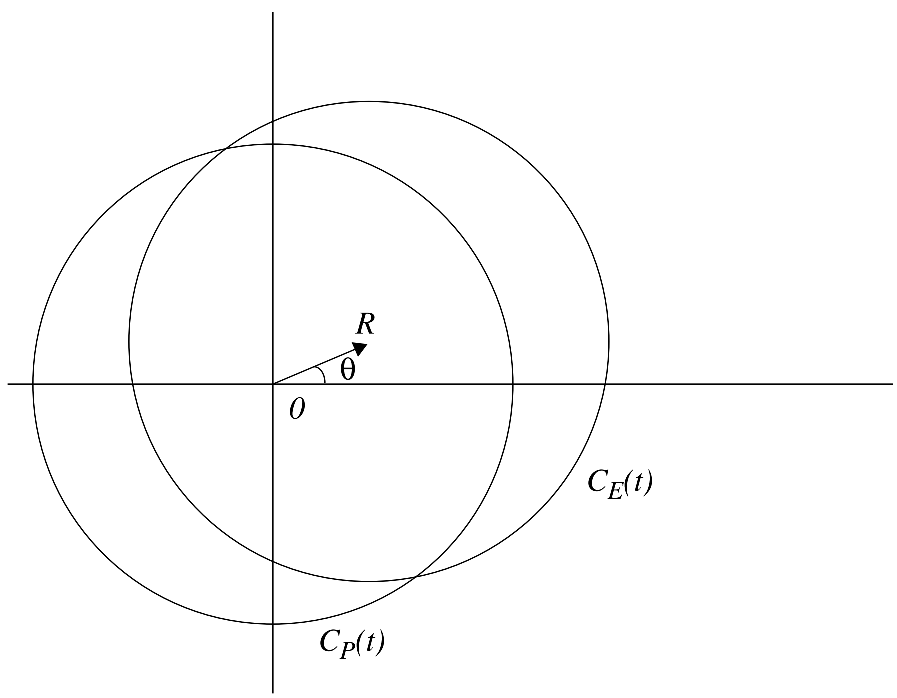


fig. 4

A possible suboptimal strategy for this case would be to accelerate in the direction of Q to minimize/maximize the percentage of the common area inside both circles. The problem with this reasoning is that the angle θ is varying with t, as $R=X_E(0)+tV_E(0)$.

Both circles have now radius $At^2/2$. Dividing by t^2 we obtain that the center R of the evader's circle is at

$$\frac{X_E(0)}{t^2} + \frac{V_E(0)}{t} \tag{7}$$

Since the first term goes to zero one order of magnitude faster than the second, (6) can be approximated as $\frac{V_E(0)}{t}$, giving an approximate value for θ that does not depend on *t*:

$$\theta \cong \arctan\left(\frac{b}{a}\right) \tag{8}$$

This is not a winning strategy for the pursuer: It just tends to equilibrate velocities but disregards distance. It will not take advantage of a suboptimal strategy on the part of the evader to reduce distances. For real applications a different, heuristic strategy will have to be used.

II: A Learning Xpilot Bot

The word bot (short for *robot*) was used at first to refer to automated software tools, sometimes with conversational capabilities, used in the Internet Relay Channel (IRC)[6]. Now the word is used in general to refer to software agents acting on computer networks through an interface that is normally used by humans. In this sense, bots are potential Turing tests for the real people using network environments.

Bots are usually easy to distinguish from humans by their behavior, which often serves some egoist purpose of their creators. So bots are often unwelcome and frequently explicitly forbidden.

Xpilot, with its client-server architecture, is a domain where computers and people can face each other in comparable, if not equal, conditions. A client program will not have access to more information than a human opponent, as the information sent by the server over the network is the minimum that the client needs to make an accurate display. Nothing is communicated that does not have a graphic expression. This characteristics make Xpilot an interesting domain for the creation of an intelligent bot.

Server robots

There are Xpilot robot players included with the standard game software. They are, however, server robots, part of the server program and based on its internal representations. They are far from facing the information constraints of a client-side player.

These robots are controlled by a long program in which answers to many situations that can take place have been written. The code resembles that of Rogomatic's expert system [4]. On this paper we will use the word "bot" to refer to our client robots, that interact with the server in the same way humans do, and "robot" for the server, expert-system style programs.

The availability of these robots can be exploited by bots and people for solitaire playing. We will use them as training sparrings for our bot.

Software considerations for an Xpilot bot

Our approach to programming an Xpilot robot involves three steps:

a) Transform the information received by the client to compute input state variables that the bot can use to take its decisions.

b) Program a set of high-level *behaviors* that are to be the atomic actions available to the bot. These behaviors will send action request commands to the server for execution.

c) Prepare a learning environment that will be executed every time frame and fed the bot's inputs. These routine will call in turn the desired behavior routines.

As in Koza's Pac-Man [5], instead of trying to apply a learning technique to the full complexity of the game, we simplify the task by *hacking* some high-level behaviors. We would like to achieve an intermediate point where the human effort put into the programming of the inputs, behaviors and learning algorithm together would be less than or equal to the expected effort of just hand-coding all the actions of the bot. As we shall see, this objective was only partially, if at all, achieved.

Bot inputs

One main problem when programming our bot is that there is no notion of the conservation of objects from one time frame to another. The server sends a list of objects, ships, obstacles, treasures, bullets, debris, etc., on every frame that completely replaces the previous one. It is difficult, for example, to estimate the velocity of a given object, since for that it has to be identified with a similar object from the previous frame.

The inputs used in the bot are:

1. Current (x,y) position of the bot
2. Current aiming angle
3. Angle difference between the current bot direction and that of the nearest enemy
4. Distance from nearest enemy
5. Number of enemy ships in close range
6. Current (x,y) position of the nearest enemy
7. Number of bullets in close range

This few inputs were coded with the hope that they could be enough for the bot to identify danger situations and favorable ones and act accordingly. A number of aspects of the full Xpilot game have not been considered, such as long-term strategies (treasure hunt), acquired abilities, shields, mines, teams, gravity, geography, etc. Our learning experiment will run in a server without these extra features, a pure space dogfighting scenario.

Behaviors

The following are the behaviors we have written for the bot:

1. Do nothing
2. Fire at nearest enemy
3. Wander (turn/accelerate at random intervals)
4. Stop (turn in direction opposite as of current movement and accelerate until observed velocity is zero).
5. Approach (accelerate in direction of nearest enemy)
6. Get_him (approach and shoot)

As the inputs, the list of behaviors seems somewhat redundant and incomplete. A pragmatic principle can be called to justify both: A comprehensive, reasonable, perfectionist list of inputs and behaviors is not what we are looking for. Just one that is rich enough for the learning algorithm to work.

Learning: Pablo's memorybase

How many times must I tell you not to knock at people's doors, Obelix?
Goscinni-Uderzo, *Le tour de Gaule d'Asterix*

A genetic algorithm seems inadequate for evolving a bot control program for several reasons. On one hand, Xpilot is a real time environment: An Xpilot game needs several minutes for trends in the changing score to become noticeable. Testing an individual in a real game in a server might take some 20 minutes. Moreover, only limited populations are possible: Even if lots of bots played against each other, more than 12 or 15 simultaneous players would saturate the server, generating lag problems.

A learning technique is desirable that would focus in learning through the experiences of a single individual interacting with its environment. We will adapt our memorybase model originally used for artificial *Tron* players [7]. In that first experience, the casebase proved useful but constrained by the delicate, time-consuming fine-tuning of its parameters.

Our memorybase is a proposed model for payoff-based (Reinforcement) learning. Instead of adjusting weights on a neural network, with the hope that adjusting the output function at some points will naturally extrapolate to others, we use reinforcement to create and maintain a casebase that hopefully will yield improving performance over time. The assumption is that what works on a certain situation is good in a local neighborhood of it.

Casebase reasoning (CBR) has been traditionally associated with computer 'reasoning', that is, dealing with logical inference domains [8]. Our memorybase resembles CBR in the way it tries to handle new, unknown situations by adapting others that in the past have been successful. CBR suggests that people are adapted to perform everyday tasks that correspond to well known cases. When a new situation comes to happen, we will try to adapt from a somewhat similar case we know. If such solution does not work, the limits of that behavior have to be noted and a new solution candidate has to be essayed.

Where genetic algorithms and also neural networks use compact representations, we propose that it is not necessary, for real-time applications, to use them. Today's computers can scan through a database of remembered cases very fast. We propose to use this storage and retrieval capabilities to create new learning algorithms. Considering the payoff or reinforcement to be a very sparse, expensive resource, it seems reasonable to think of learning algorithms that will remember past events and they related payoffs to construct the payoff-efficient learning paradigm that is needed in such domains.

Memorybase Algorithm

Our memorybase algorithm captures some, but not all of our expectations for a learning technique adequate for Xpilot.

1. Define a distance in the input parameter space

As we desire to adapt remembered solutions to new, similar cases, a notion of similarity is needed. If we want to set limits to the extent of a case validity, this notion has to be a measure.

Our simple solution is to create an euclidean metric on the input space by defining a distance

$$d(x, y) = \left(\sum_i (x_i - y_i)^2\right)^{\frac{1}{2}} \tag{9}$$

to measure the similarity of two world situations, that is, two vectors of input values, $(x_1,...,x_n)$ and $(y_1,...,x_n)$.

2. The memorybase will be a list of records, each consisting of

- An input value set
- An associated action
- A radius of applicability

The input value set has been stored from a previous experience. That is, every record on the memorybase corresponds to a precise memory of an event that happened in the past. The action associated with it is the one that previous experience suggested in similar situations. A radius of applicability indicates the estimated maximum difference -in terms of the distance (9)- for this same solution to be applicable.

3. Every time frame the memorybase will suggest the action to be taken

At every time frame a set of input values is read and fed to the agent control routine. This routine will search through the memorybase for a remembered case such as the current input vector is within radius of applicability.

The associated action is taken and the use of this case registered for future evaluation.

4. Payoff, when received, is used to recalculate the memorybase radiuses

An *activation status* is maintained for every case in the memorybase that estimates the part of responsibility that the recent use of a case has in the current state. This activation status is increased by one every time a case is used and multiplied by a temporal difference rate $0 < \tau < 1$

at each time step.

When a new payoff value, positive or negative, is feeded to the casebase, all radiuses are multiplied by its activation value times the new payoff. In this way, negative payoffs decrease radiuses and positive payoffs increase them.

5. Out of reach situations generate new cases

The union of all the spheres centered at each case with their radiuses covers at each time part of the whole input space. When a situation comes that is outside all of them, that indicates that we do not have a suggested behavior for that case. Either it is away from past events or our estimated applicability of similar experiences does not reach it.

In this case a new record is added to the memorybase:

- The current input vector is stored in the new case
- The initial radius is set at some predefined constant
- A random action is associated with it

6. Will the memorybase learn?

The memorybase is empty at the beginning of the learning experience. New cases are generated for the first unknown experiences as random actions are taken.

As payoff is processed, erroneous behaviors will decrease their radiuses and eventually disappear, but useful ones will increase and establish themselves.

In the previous Tron experiments we used an aging factor to force the radiuses to converge towards a final value. This meant that the casebase ceased to adapt after some time. Now we are leaving this factor out to produce an ever-adapting memorybase. This means that optimal behavior will never be achieved. Instead, the memorybase will reach an oscillating equilibrium state where positive payoffs increase radiuses past their optimal value, generating new negative payoffs that will reduce them back.

7. Partial event scope

Finally, we think that the sphere model limits the possible extension of a case to cover its natural domain. It might be important, for example, to associate a situation with a lot of near enemies with an escape behavior. This behavior would be independent of the current (x,y) coordinates.

We thus added flexibility to the casebase by generating not only a random action to couple to each new case, but a random subset of inputs to be used for that case. This means that not all of the terms $(x_i\text{-}y_i)$ will be computed in (9), only those that were marked as applicable.

Experimental setup

After various initial trials that served to test the various behaviors and adjust the parameters of the memorybase, an experimental setup was fixed with eight robots where the bot would enter the game and try to learn how to get a decent score. Human, fixed and random behavior runs were done for comparison.

The casebase parameters were set as follows:

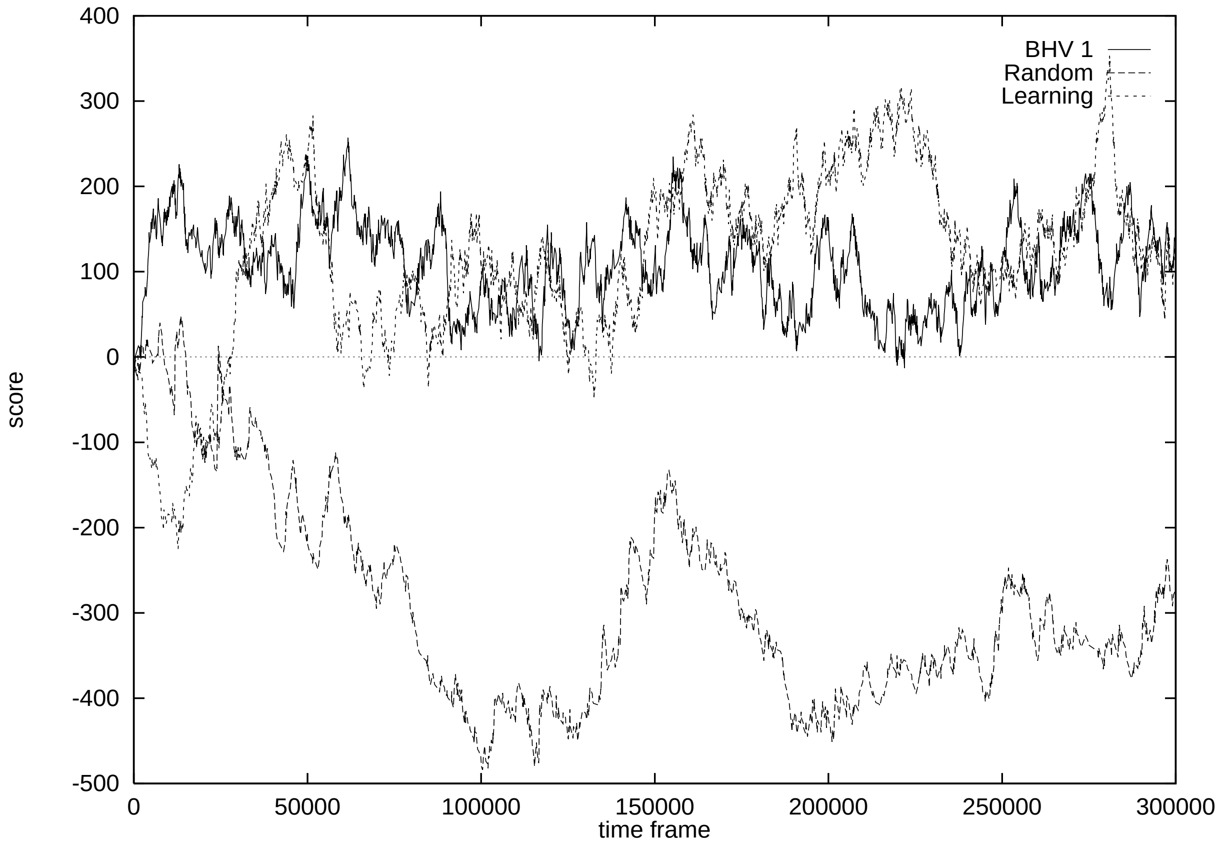
NEW_RADIUS = 100. The initial radius assigned to newly created cases.

EXPONENT = 0.98. The activation decreasing rate τ per time frame.

PAYOFF_MULT = 5. The raw Xpilot scores where multiplied by five.

Results

The results of our experiments were not conclusive. On one hand, the learning bot had a score far away better than a random bot generated by randomly switching behaviors. This means that the casebase algorithm was successful in assigning temporal credit to positive and negative payoffs. On the other hand, however, the resulting casebase was dominated by a single behavior: “Fire at nearest enemy”.



graph 1

As illustrated by graph 1, the bot score is negative in the beginning, resulting from its random behavior. Soon, however, it improves, and by time step 50000 it is mainly using behavior 1 (fire!). The score of a player programmed to use behavior one all the time is also plotted for comparison.

Even a random player obtains positive scores from time to time. This is fundamental for learning to happen. If random actions never yield positive payoffs, no improving is possible.

It must be noted that graph 1 reflects an important property of the way Xpilot handles scores. When a player has positive score, it receives smaller positive scores for killing enemies, and higher negative scores for being killed. This means that over time the score of a player will fluctuate around an average value, indicating its overall performance, or kill/die ratio.

The average score for the memorybase player and pure behavior 1 are very similar, 116 and 107 respectively, versus -289 for the random player. For comparison, a human player might score an average of 76 points and the score of the server robots is around zero.

III: CONCLUSIONS

Xpilot proved to be an interesting domain for artificial opponents research, as it shows difficult yet tractable features as an idealized differential game, as well as a domain for programming a client bot to participate in the game as humans do. It was shown that the temporal credit assignment can be solved in this game as it is possible to relate the raw scores with good or bad actions taken in the recent past.

Dynamic simulation games are an interesting field for artificial intelligence research because they have both challenging features as found in traditional board games and dynamic characteristics like those found in physical domains.

The differential game was an interesting exercise but its results were of little practical use. Humans themselves find it difficult to approach a wandering ship by using the rotate and thrust controls. The server robots can get caught in eternal orbits, without being able to reach a motionless opponent. An effective strategy that handles frictionless pursue against non-optimal evaders is yet to be found.

When mathematical models are taken into practice, several difficulties arise in dynamic real-time domains, even virtual ones. The velocity of an object for instance can never be calculated, as its mere conservation between one time frame and the next one is difficult to estimate. This problem suggests studying a question that was not addressed in this work: How can conservation of objects and temporal properties (such as speed or acceleration) be learned? How does it take place in the human mind?

Payoff-efficient algorithms are needed to handle the real-time constraints of dynamic domains. The memorybase learning model intuitively has interesting properties but some theoretical understanding is necessary. An effective methodology for computer agents might be to store actions and payoffs in a memory database for further analysis. We have tried to take a step in that direction.

The memorybase proved to be capable of finding a successful game strategy, but unexpectedly it consisted of a fixed constant behavior: Shoot all the time at the nearest enemy. It is reasonable to think that this is the correct solution as for most situations in our scenario the best choice was shooting, even when surrounded by enemies. But it is also possible that our memorybase simply was not capable of finding mixed strategies.

There is a lot of work that can be done to improve our Xpilot bot, but a research program should study first the general properties of this learning method, testing it in simpler domains were the results can be more accurately measured, and comparing it to other techniques.

References

[1] Isaacs, Rufus. *Differential games; a mathematical theory with applications to warfare and pursuit, control and optimization*. New York, Wiley, 1965.

[2] Lewin, Joseph. *Differential games: theory and methods for solving game problems with singular surfaces*. London; New York: Springer-Verlag, 1994.

[3] Symbolic Computation Group, *Maple version 4.2,* University of Waterloo, Dec. 1987.

[4] Mauldin, M.L., Jacobson, G.J., Appel, A.W., and Hamey, L. G.C. Rogomatic: A belligerent expert system. *Proceedings of the Fifth National Conference of the Canadian Society for Computational Studies of Intelligence*, 1984.

[5] Koza, J. *Genetic Programming*. MIT Press, Cambridge, 1992.

[6] Leonard, Andrew. Bots are Hot, *Wired* 4-04, 1996.

[7] Funes, Pablo. Using (some sort of) CBR to drive intelligent Tron players. 1995.

[8] Kolodner, Janet L. An introduction to Case-Base-Reasoning. *Artificial Intelligence Review* 6, 1992.

[9] Tesauro, G. Practical issues in temporal difference learning. *Machine Learning*, 1992.