World Automation Congress © 2010 TSI Press.

THE XPILOT-AI ENVIRONMENT

GARY B. PARKER AND DAVID A. ARROYO
Computer Science Department
Connecticut College
New London, CT, USA
{parker, darroyo}@conncoll.edu

ABSTRACT -- AI researchers face the challenge of obtaining sufficient test data to evaluate new systems for learning agent control. There are various factors such as cost and long development cycles which can dampen the progress of AI research when testing methods on actual robots. Methods for autonomous agent learning can be tested in interactive game environments as long as control of individual agents can be enabled and there is a reasonable means of testing the success of the agent controllers. Xpilot-AI is introduced as an effective testbed for rapid development of intelligent control systems. Xpilot-AI provides an easy to use, inexpensive and flexible test environment.

Key Words: Xpilot, game, space, combat

1. INTRODUCTION

AI researchers test the limits of computers' capabilities, building complex systems that challenge popular notions about the intelligence a computer can possess. To that end, they endeavor to design intelligent systems which are often radically different from anything available on the market. However, the benefit of new techniques is often difficult to measure. More problems are faced when trying new AI techniques in robotics. The robots can be expensive to build, maintain and run.

Computer games present an attractive alternative for AI research. AI researcher John Laird writes [1], "By working in simulation, researchers interested in human-level AI can concentrate on cognitive capabilities and finesse many of the pesky issues of using real sensor and real motor systems. . . ." Games are useful for their flexibility; the virtual environment may be customized to have unique physical laws or other features that reveal strengths and weaknesses of the system being tested. Making changes is easy and does not require purchase of new materials. It is also easy to pit an intelligent system against a human opponent, and gather qualitative and anecdotal data on the effectiveness of the system. Moreover, Laird states, computer games present a unique opportunity to create a more general-purpose human level AI that competes directly with humans.

Xpilot-AI is a set of modifications to the classic space combat game Xpilot. These modifications allow AI researchers to develop agents that can play against each other and humans on an Xpilot server. A large number of server options allow the environment and game rules to be customized, and agents are developed quickly and interactively. The Xpilot-AI environment was designed as a testbed to quickly evaluate the effectiveness of AI techniques by providing a flexible environment with a fast development cycle.

Figure 1: A player navigates his ship through an Xpilot map. A missile upgrade is visible in the top-left corner.

2. XPILOT

The classic computer game Xpilot is a 2D space combat game with online multiplayer capabilities. Figure 1 shows a small example screen shot of the graphical Xpilot client. Players navigate their ships through various maps, avoiding traps and collecting power ups. There are several game modes available, such as the common deathmatch mode, where players try to eliminate other ships. There are also team-oriented gameplay modes, such as a capture the flag variant, and racing modes.

Xpilot servers can send information to the Xpilot meta server located at `meta.xpilot.org`, through which clients can obtain a list of active servers. Control of the ship in Xpilot is done through the keyboard or mouse, with the three primary actions being to turn, thrust, and fire bullets. There are several additional commands that control secondary items such as lasers, missiles, shields, and tractor beams.

In Xpilot, physical forces, such as inertia, friction, and gravity are modeled realistically. Ships have mass, and applying the correct direction and amount of force to navigate them through the environment requires skill. Often it is walls that present the greatest threat, and designing a program to avoid wall collisions intelligently is a challenging research objective. Xpilot maps may contain cannons which fire at players of their own accord, wormholes that warp players from one section of the map to another, gravity wells that change the trajectory of flying objects, and several other features that make the environment interesting.

Xpilot was chosen as an environment for AI research for several reasons. The variety of game play modes and server options give researchers great freedom in designing the tests for intelligent systems, and allow for agents to be evaluated in many different ways. The low system requirements to run the game allow for dozens of agents to be run simultaneously on modest hardware, making it a useful platform for the study of team dynamics. A large variety of different maps are available, and a map editor is available for creating custom maps.

The code base for Xpilot is written in C. Xpilot is open source, and freely available. The ability to modify the source code allows researchers to make large changes in the environment that would not be possible otherwise. There exist projects like Xpilot-AI that are built around other existing open source games, such as the QASE project for Quake 2 [2]. While QASE is Java based, Xpilot-AI is written in C and has language bindings for Java, Python, and Scheme.

3. XPILOT-AI

Xpilot-AI is a library written in C that allows researchers to easily develop Xpilot agents that control in-game ships on the Xpilot server. Xpilot-AI provides several functions through which agent control programs are able to obtain information about the environment around them, the ships on the map, and various other useful data. Foreign bindings exist for Java, Scheme, and Python which allow agent control programs to be written in any of these three languages. Using the Scheme or Python bindings, the agent can be developed while it is running in the read-eval-print loop.

Xpilot-AI is implemented as a set of modifications to the C source code of the original Xpilot client. Xpilot client and server stay synchronized on a frame-by-frame basis, sending packets conforming to the Xpilot protocol. The server sends packets containing information about changes in the environment, and the client sends packets containing information about key presses and other actions performed by the player. Xpilot-AI simply parses the data received from the server and creates packets to send from the client.

The functions exported by the Xpilot-AI library perform actions by simulating key presses to the Xpilot client. Information about the map and surrounding objects is gathered from the drawing functions of the Xpilot client. There also exist functions to send and receive messages through the Xpilot chat system. This allows separate in-game agents to communicate with each other and the Xpilot server. Xpilot-AI also provides the function `AImain`, an empty function that is called for every frame of execution. Agent

```
void main(){
    AI_xpilot_launch();
}
void AImain(void){
    AIself_turn(20);
}
```

Figure 2: A simple C program to launch a bot that turns 20 degrees counter-clockwise every frame.

programs are written by redefining this function. The C program in Figure 2 is an excerpt of a controller program that turns the agent 20 degrees counter-clockwise every frame.

The C code in Figure 5, once compiled and executed, will open an Xpilot client window, from which the user may choose a server to join. Once a server is chosen and play begins, the function `AImain` will be called once at the end of each frame. The definition of `AImain` above will cause the agent ship to turn 20 degrees counter-clockwise every frame.

One of the strengths of Xpilot-AI is that it allows researchers to pit different types of autonomous agents against one another in competition. Agent controllers can be written in different languages, run from anywhere in the world with internet access, and use different learning methods in their development. Comparison of different methods is easily accomplished by putting the agents in the same arena. As an example of the ease in comparing two independent agents, the next section describes a competition between the default bot that is part of the Xpilot server code, and a custom agent written using Xpilot-AI.

4. EXAMPLE: COMPETITION BETWEEN TWO AGENTS

4.1 The competition

Several matches between the bots written as part of the Xpilot server code and custom bots written by researchers using the Xpilot-AI library were conducted. For the first test, kill rate is considered the metric of success for an agent. Other useful criteria that can be used to judge an agent include crash rate and longevity (the length of time an agent survives without being killed or crashing). In addition, the Xpilot game includes methods for keeping score of the opponents. Two of these methods were also used to compare the bots discussed in the next section.

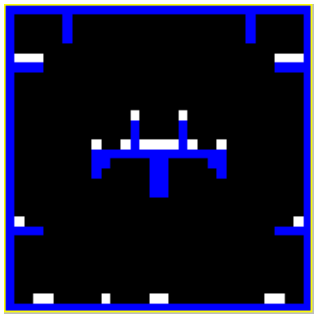

Figure 3: The Xpilot map "Lifeless." Obstacles are in blue, starting positions are white.

For the competition, the Xpilot map "Lifeless" was used. It is a simple square map, walled off on all sides, with an obstacle in the center which may be used for cover. An image of the map is shown in Figure 3. When a player is killed, he respawns at a random starting position. Both players have an unlimited number of lives, and there are no rounds. Statistics were sampled every 5 seconds of gameplay.

4.2 The opponents

The two opponents in the first test were Morton, an expert system written by the Xpilot-AI developers, and the default Xpilot server bot. Morton is a simple rule-based program written as an example of a non-trivial agent using the Xpilot-AI library. The program contains rules to avoid walls, pursue and attack enemy players, and dodge shots. The version of Morton used consists of about 100 lines of C source code. Versions of Morton written in C, Java, Python, and Scheme are distributed with the Xpilot-AI code base, which is available online at `www.xpilot-ai.org`. For the second test, an extra agent, Sel, was introduced. Sel is a rule-based system, like Morton, written using the Scheme bindings for Xpilot-AI. Sel has been optimized for competitive play, and is currently the best hand-coded rule-based control program for Xpilot-AI.

4.3 Results

The results for the first test were decisive; the Xpilot-AI agent, Morton, was consistently able to eliminate the default Xpilot bot without being eliminated. A graph of the kill rates of the two bots is shown in Figure 4.

Figure 4. The number of kills for each agent, sampled every 5 seconds.

Many different metrics of measurement are possible with the Xpilot-AI environment. Using kills compares the agent's ability to kill the other agent while avoiding being killed, but says nothing about its ability to avoid dying by colliding with walls. Comparing agents by their score, on the other hand, gives an indication of the agent's ability to eliminate other ships, but also its ability to avoid crashing into walls. Measurement by game score is also useful in other game modes, such as capture the flag, where the ultimate objective of the game is not elimination of other ships.

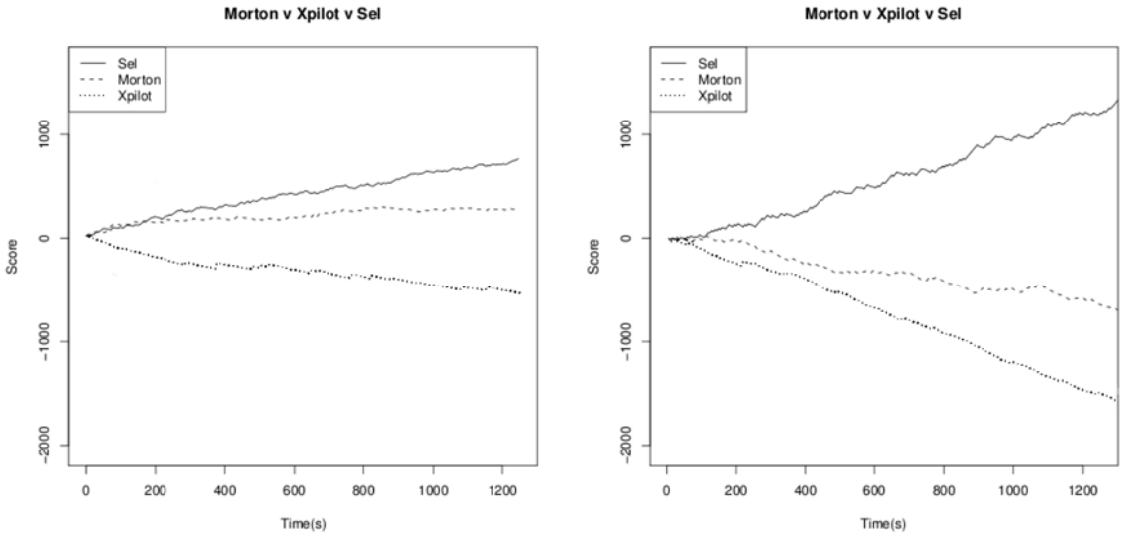

Figure 5: Comparison of the two scoring methods. The first graph uses data gathered using using Xpilot's scaling algorithm that changes the kill reward to keep the game close. The second set was created using constant scoring.

The Xpilot scoring system is highly customizable through command-line flags in the Xpilot server program. By default, the reward received for eliminating an enemy is not constant, but dependent on the scores of both the agent and the enemy ship that was eliminated. Eliminating a ship with a higher score earns a larger reward that eliminating a ship with a lower score. This system was designed to keep games between human players close. The default scoring method may be disabled in favor of a constant reward regardless of score by supplying the `constantScoring` flag to the server command. The graph in Figure 5 shows the results of the second test using these two scoring methods. In the second test, all three agents (Sel, Morton, and the Xpilot-provided agent) were put in the arena at the same time. The scores of the agents over several frames are shown in Figure 5. As can be seen, with the standard scoring method designed to keep the game close, the difference in the two agents is not as stark. For this reason, the constant scoring flag is probably desirable during agent training where similar agents are being compared.

Researchers may customize the effect an event has on the player's score through the use of command-line flags to set score multipliers for each event. For example, if a researcher were using the score as feedback to train an agent to avoid walls, he may set the multiplier for the crash penalty to a very high value while decreasing the other multipliers. This gives researchers great flexibility in training their agents to learn what is most important for the task they are designed to perform.

The marked difference in performance of the Xpilot agent versus Morton, shown in Figure 4, is telling of the different objectives and constraints between the two agents--the default Xpilot bot runs as part of the server process, and must share resources with it. Making the agent too computationally expensive would cause gameplay to slow down in the presence of several bots. The Xpilot-AI agent Morton, is under no such constraints and has more room to make expensive calculations. The significant difference in the scores of Sel versus Morton shows how improvements in the control program will be evident in the performance of the agent (Figure 5).

The Xpilot-AI environment has been used to test controllers using a variety of techniques, including rule-based systems such as Morton, neural networks [3], and evolutionary computation [4]. Researchers can decrease the time required to test controllers by increasing the frame rate of the game, or by distributing evaluation across multiple agents to run the tests concurrently [5].

5. CONCLUSION

Xpilot-AI is a useful environment for evaluating new designs of artificially intelligent systems. The flexibility in environmental settings, online multiplayer capability, ability to pit different types of agents against one another, and the rapid development cycle contribute to form an excellent tool for AI research. Xpilot-AI gives researchers the opportunity to test and compare control systems and learning methods before using them for more costly applications.

ACKNOWLEDGEMENTS

This research was supported in part by the Andrew W. Mellon Foundation through a grant to the Connecticut-Trinity-Wesleyan Computer Science Consortium.

REFERENCES

1. John E. Laird and Michael van Lent. 2000. “Human-level ai's killer application: Interactive computer games.”
2. Bernard Gorman, Martin Fredriksson, and Mark Humphrys. 2005. “QASE: An integrated api for imitation and general ai research in commercial computer games.”
3. Matt Parker and Gary B. Parker. 2007. “The evolution of multi-layer neural networks for the control of xpilot agents,” *The IEEE Symposium on Computational Intelligience and Games (CIG 2007).*
4. Gary B. Parker and Matt Parker. 2007. “Evolving parameters for xpilot combat agents,” *IEE Symposium on Computational Intelligence and Games (CIG 2007).*
5. Matt Parker and Gary B. Parker. 2006. “Using a queue genetic algorithm to evolve xpilot control strategies on a distributed system,” The 2006 Congress on Evolutionary Computation (*CEC 2006*).