World Automation Congress © 2010 TSI Press.

REAL-TIME AI IN XPILOT USING REINFORCEMENT LEARNING

MARTIN ALLEN, KRISTEN DIRMAIER & GARY PARKER
Computer Science Department,
Connecticut College,
New London, CT, USA

ABSTRACT—Reinforcement learning (RL) allows agents to learn a best-possible long-term course of action, based on immediate positive and negative rewards. This approach enables real-time learning, since the agent constantly adjusts the value of actions taken, eventually selecting that action with highest value in each environment-state it encounters. We investigate the use of the Q-learning RL technique in an agent that learns to intelligently navigate the Xpilot video game environment in real time. We compare learning performance for different reward and action models, and discuss the challenges of RL methods in such a reasonably complex domain.

Key Words: Reinforcement Learning, Real-time AI, Games, Xpilot

1. INTRODUCTION

Computer and video games are rapidly increasing in quality and complexity. As a result, there is particular interest in the use of artificial intelligence (AI) to control game play. A major goal of such work is allowing players to interact with intelligent game agents in real time. This requires the agents to be able to change behavior or strategy based on interactions with their opponent or environment as they occur. A useful test bed for game AI is the computer game Xpilot, in which player-agents control "spaceships" in a simulated combat environment. The ultimate goal of our research is to develop a cooperative team of Xpilot agents that learn to defeat enemies in real time. This paper discusses stage one of that process, which is to create an agent that learns to intelligently roam around the space without crashing fatally into walls.

Incorporating real-time planning into AI game strategy has many challenges. Baumgarten et al. [1] approach real-time learning in the DEFCON computer game by combining simulated annealing methods, decision-tree learning and case-based reasoning. Stanley et al. [2], working with the NERO game, employ an evolutionary neural approach, starting with simple networks and adding nodes and connections as the game is played, allowing agents to evolve increasingly sophisticated behaviors in real time.

While these approaches allow for real-time learning, in both cases agents select strategies based on predefined plans, with adjustments made after a game is completed. In our approach, on the other hand, we develop an AI agent that learns and adjusts during actual game-play. We employ reinforcement learning (RL), in which agents update long-term action-value functions based on short-term costs and rewards, and work towards optimal policies [3]. In particular, we adopt the Q-learning approach, whereby agents learn values for particular state-action pairs, based on outcomes in a stochastic environment [4].

An early application of RL methods to real-time control of a navigational agent is that of Madden and Nolan [5], who use tabular Q-learning for automated vehicles. In the game domain, Sharma et al. [6] use a case-based reinforcement learner (CARL) to learn strategies in the MadRTS video game. Neither approach employs RL in an environment as complex as Xpilot, however. We next introduce the Xpilot game and research platform version, Xpilot-AI. We then describe our RL approach, with results and conclusions so far, based on two different models of behavioral reward. We conclude with notes on future work.

2. XPILOT AND XPILOT-AI

Xpilot [7] is an open-source 2-D space combat game, similar in style to Atari Inc.'s game, Asteroids. It can be played over a network or via the internet. Players control ships with available actions like shooting, thrusting and turning. The object of the game is to obtain points by shooting and destroying enemy ships. One major challenge is the frictionless environment and the life-like physics of the game. As well as trying to avoid fire from enemy ships, the player must also avoid hitting the walls at high speeds. Backfire from killing an enemy can also force the player's ship into a wall. Thus, in combination with a large set of state variables, Xpilot is an extremely challenging test-bed for both offline planning and real-time learning [8].

Like most computer games, Xpilot comes with pre-programmed artificially intelligent agents to play against. However, it also has a versatile programming interface, Xpilot-AI, allowing researchers to develop their own AI agents, to compete against other computer-controlled agents or human players. In our work,

we used Xpilot-AI to experiment with reinforcement learning control of an automated ship-agent. We employed a predefined game map, Simple, which is a relatively large open field. Our initial objective was to develop and RL agent that could learn to intelligently roam around this map without running into walls.

3. THE Q-LEARNING APPROACH

In reinforcement learning (RL), agents acquire and improve skills based on immediate rewards, both positive and negative, when performing actions in a given environment. Over time, such learning methods seek guide polices of action in order to maximize long-term reward. The learner is not told which actions to take, but instead must discover those yielding highest reward, by trying them repeatedly over time. Thus, the RL learning model generally consists of the following components: (i) an agent, (ii) a set of actions, (iii) a state space, (iv) a set of rewards received for taking actions in a given state, and (v) a policy of action, adjusted over time. Assuming that the agent can observe the current system state directly, the goal is to learn a policy (a function from states *s* to action-choices) that maximizes long-term expected reward.

The particular technique used here is called *Q*-learning [4]. In this approach, the agent uses immediate rewards to adjust a value function, *Q(s, a)*, which assigns a particular numeric score to given state-action pairs (*s, a*). When not exploring alternative courses of action, the agent follows the best policy learned so far, simply by observing the immediate state *s*, and choosing the action *a* such that *Q(s, a)* is maximized. The agent initially has no information about the environment. It begins by taking random actions and receiving rewards for those actions. Eventually, by adjusting the *Q*-value function based on those rewards, it learns to take the actions that yield the true highest value for that state, given expected future states.

One strength of this sort of approach is that it only updates values and policies based on the actual states and rewards it actually happens to encounter. Thus, it is able to compare the expected utility of the available actions without always requiring a model of the entire environment. This is useful when the environment is highly complex, as it is in a game like Xpilot.

4. REINFORCEMENT LEARNING IN XPILOT

In some ways, Xpilot is ideally suited for a reinforcement learning approach. The game is frame-based, so that each moment of play corresponds to an individual state of the overall system. The player (human or AI) is polled at each frame for an action selection, and the resulting next frame is calculated based on that action and the relatively complex underlying game physics. The challenge, from the point of view of our research, is to choose a state representation that simplifies the highly complex game dynamics, and to select a possible set of actions over which the agent will learn a policy. Further, we must assign a state-by-state (frame- by-frame) reward function to action outcomes, in a way that reinforces the behavior we want our agents to learn, namely safe navigation through the game map. Furthermore, we want our agent to end up moving realistically, in the sense that it behaves somewhat like what we would expect from an intelligent human player, moving back and forth across the screen while avoiding crashes into walls.

4.1 States

One of the most difficult aspects of reinforcement learning is abstracting the state space. In order to learn successfully, the agent's state space must be large enough to properly represent important features of the environment, containing enough variables (and the right ones) to distinguish between environment states for which the value-maximizing output policy will be relevantly distinct. At the same time, since RL works only via repeated reinforcement based on immediate rewards—especially in the presence of noisy, stochastic state transitions—this state-space must not be so large that the agent cannot visit all of the states several times, or cannot easily compute, store and access its *Q*-value function. Unfortunately, in its basic form, Xpilot-AI contains so many state variables, all readily available to an agent if so desired, that its full state-space is far too large to expect effective learning to be feasible.

Thus, the state space we chose to use was made up of 384 states. The state variables and their possible values can be seen in Table I. The current state information was stored in an array of integer values, and consisted of a single 6-valued variable to keep track of the ship's range of speed (from not moving to very rapid), 4 binary variables to track the ship's location with respect to the map's walls, and 2 binary variables to track if the agent is alive and whether or not it has recently crashed. The Speed variable was separated into 6 discrete ranges, with values from 0 to 35, corresponding to different possible ship speeds within the Xpilot game; thus, for instance, if the Xpilot ship was travelling at any Xpilot game speed

State Variable	Range (Xpilot)	Value (RL)	State Variable	Range (Xpilot)	Value (RL)
Speed	0	0	Near East Wall	N/A	0/1
	1–4	5	Near West Wall	N/A	0/1
	5–9	10	Near North Wall	N/A	0/1
	10–14	15	Near South Wall	N/A	0/1
	15–20	20	Alive in Prior State	N/A	0/1
	> 20	35	Alive in Current State	N/A	0/1

Table I. The state space is made up of 384 states, with 7 possible state variables. ``Range" indicates the possible value proper to Xpilot itself (if defined), while ``Value" is how that variable is represented in the RL agent's own representation of the environment

from 5–9 inclusive, the agent would represent this fact by setting its Speed variable to value 10. It is also important to note that the binary *Near Wall* variables are set as a function of the speed. The faster the agent is moving, the further ahead it checks to determine whether or not it is near a wall, and the further it can be from a wall and still set the corresponding variable to True (1). This is to make sure that the agent has enough time to turn away from the wall before crashing into it. Also, we note that it is possible, particularly in corners of the arena, to have more than one *Near Wall* variable set to True.

4.2 Actions

Again, the full set of Xpilot game actions was somewhat too large for our purposes. In addition, they were often simpler than desired, and were supplemented by compound combinations of atomic actions. Thus, we used a set of 19 complex actions that included turning in one of eight possible cardinal directions, turning thrust on or off, or combining a turn with a thrust; we also included a "do-nothing" (NOP) action.

As indicated in the previous section, the agent does not keep track of its heading or tracking, i.e., the direction in which the ship's nose is pointed or the direction of its flight path; thus, the Turn action in our RL experiments calculates the heading and number of degrees needed to turn in order to point in the one of the eight cardinal directions—where North represents the wall on the top of the screen—and returns the corresponding Xpilot game command to properly execute the desired rotation. For example, if the chosen action is Turn North, the agent calculates its current heading and uses this value to determine the number of degrees needed be facing up toward the top of the screen. We note that as things stand now, there is no maximum value set to the number of degrees a ship can rotate in a single frame; each turn thus executes instantly, something that is not always the case in human-guided play, where Xpilot runtime options can be set to restrict turning to something like 15° at a time in any direction.

In order to ensure that our state space and action choices were indeed useful bases for a learning agent, we hand-designed a deterministic-control test agent that acted on a predetermined set of Q-values for each action in a given state, without ever changing those values. The Q-values, in turn, were based on the actions we felt the agent should take in each state, based on play-testing of the game mechanics. For example, it would seem logical that if the agent were flying directly at the North wall the best action would be to turn South and thrust to avoid crashing. Thus, when the agent was in the state s = *Near North Wall*, we assigned the action a = *Turn South and Thrust* a positive Q-value, and gave all other actions a value of 0. As it turned out, our hand-coded agent was able to sustain indefinitely long flights around the game arena, thrusting and turning constantly to maintain a safe distance from walls, turning out of corners properly, and generally mimicking a competent human player. We were thus able to show that the state space and action set were in principle descriptive enough for an agent that could learn how to navigate the space effectively.

4.3 Rewards

The final and arguably most important part of reinforcement learning is the reward function, since RL agents work to maximize the expected total of this value, with no other marker of "success" or "failure" in learning. Thus, the reward function in an RL domain has to be chosen carefully, so that the maximal value policy an agent actually learns corresponds to the intelligent behaviors expected in, and appropriate to, that domain. Often, this is a non-trivial endeavor, especially when system dynamics are complicated, or outcomes of actions are difficult to evaluate on a relative numerical scale.

Both R1 & R2	
Reward Value	**Condition**
-1000	Agent crashes
0	Agent is not moving
$+12$	Agent is near 2 walls
$+25$	Agent is near only 1 wall
$+50$	Agent is near 0 walls

R2 Only	
Reward Value	**Condition**
-5	Agent takes action other than thrust or do nothing
$0.2 \times \texttt{Velocity}$	Agent is alive

Table II. Reward values for the two structures, *R1* and *R2*.

We tested several reward functions. The two that form the basis of the results in this paper we will call *R1* and *R2* (Table II). The first of these reward-structures, R1, includes five possible conditions:

- The agent crashes into a wall; the large negative penalty of $r = -1000$ reflects how undesirable that is.
- The agent is not moving at all; the reward $r = 0$, so the agent can not achieve a value-maximizing policy by simply floating safely in its starting position on the game screen
- The agent is near one or more walls; here, reward decreases as number of walls increases, reflecting increased danger. Note that if an agent is not moving, then the speed-based Near Wall state variable is always False; thus, motionless agents gain no reward, no matter their relative position on the map.

The second reward-structure, R2, includes everything given by R1, with two new possibilities.

- To dissuade agents from rotating constantly (which had no effect on original reward-structure, R1), a penalty of $r = -5$ was added for any action other than thrusting or doing nothing; this is an example of the use of the reward function to generate "natural" and "intelligent" behavior, since a human player will not usually flip the ship about uselessly as it flies along its path.
- To encourage agents not simply to fly very slowly in an effort to avoid crashing (at low enough speeds, a ship can bounce off a wall unharmed), a reward of 0.2 times its Xpilot velocity was added.

At each frame of the game, an agent's last state, s, and action, a, are evaluated based on current conditions, as given in the relevant reward-table. For each reward-condition that is true, the given value is added to total reward. This sum is the agent's final reward, r, for the Q-value update equation.

5. RESULTS AND ANALYSIS

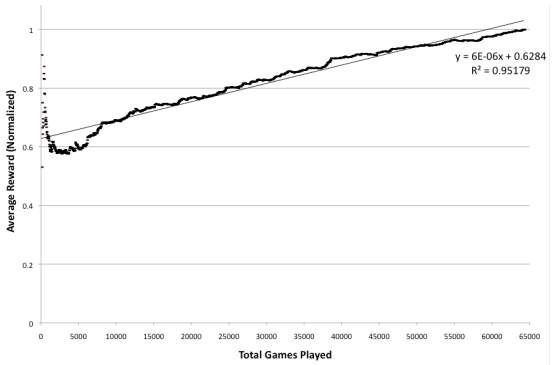
Using the state and action sets outlined, in combination with the two reward-structures described, we tested a pair of distinct agents, running them for several tens of thousands of Xpilot games each. We now describe our results, both in terms of qualitative fit to the idea of "natural" or "intelligent" navigation, and in terms of quantitative reward-maximization.

5.1. Qualitative Results

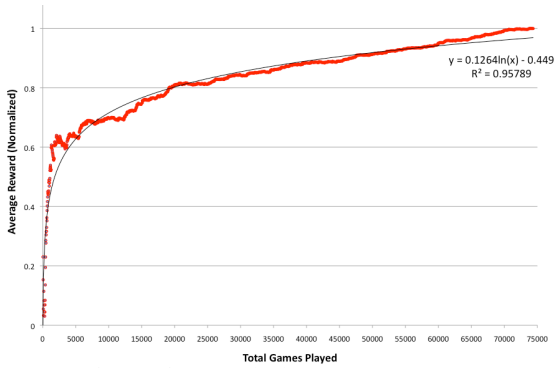
We observed three main things in our testing. The first, as already described, was that the relatively simple state space and action set chosen were well enough designed to learn the desired policy. This is shown by the fact that the deterministic test agent we designed with predefined Q-values successfully fly around the space without crashing. Thus, the main goal of our research became deriving a reward function so learning agents could converge on a set of similar Q-values. This proved significantly challenging.

With both reward functions, some learning was clearly observed. Our second significant observation involved the R1 reward-system, for which two main agent strategies developed. While neither approach fit our intuitive sense of "natural" or "intelligent" flight, each was effective in helping maximize reward and navigate somewhat successfully. The first such technique is that the agent would constantly flip 180◦ back and forth through the middle of the map, thrusting once in each direction to slow itself down. Then, when it approached the walls (based on its speed-relative Near Wall state function), it would stop thrusting and simply glide. The other observed technique involved thrusting once and then simply floating around in the space at a very slow speed. This allowed it to bounce off of walls and stay alive through many frames.

The third significant observation was that reward-system R2 helped eliminate the flipping behavior. Given that set of rewards, the agent would approach walls at slow speed, before making a single turn to move in the opposite direction. While the results of this approach were largely better, the slow speed would need to be addressed if the agent were expected to engage in actual competitive game-play.



(a) *R1*: positive, primarily linear accumulation of average reward, showing steady learning, continuing even after 64,250 games.



(b) *R2*: log-form gain in average reward (more typical in RL), showing initial burst of learning, with an apparent plateau near 74,350 games.

Figure 1. Learning performance: average normalized reward gained per game over time for *R1* & *R2*.

5.2 Quantitative Results

More specific results for each reward structure can be found in Figure 1, showing average normalized reward over time, plotted against the number of Xpilot games played by the learning agent. For purposes of measuring total accumulated reward, an episode terminated as soon as the agent died by hitting a wall, and the next game did not begin until the agent began moving once again. Total reward is averaged at each point over games to date; furthermore, the result is normalized, as the particular value only distracts from the trend (since, for instance, the same RL agent can learn the same policy, but gain higher or lower average absolute reward, simply by scaling each *r*-value uniformly by some constant multiplicative factor).

In the case of *R1*, in 1(a), the result is a strikingly linear trend. While the average is somewhat noisy over the first few thousand games (as to be expected given the smaller sample size), it soon stabilizes, and by 10,000 games has settled into a significant steady upward trend, still climbing at the experimental limit of 64,250 games (roughly 24 hours of compute time). This is striking, showing that the agent still has room to learn even very deep into game play, suggesting both the complex dynamics of even the simplified state- and action-space for Xpilot, and that even longer-term behavior needs to be analyzed in future.

Results for *R2*, in 1(b), are somewhat more typical of those seen in many RL domains. In particular, the log-form trend given by regression produces a common RL performance profile, whereby the agent learns rapidly at first, quickly gravitating toward a better policy, but begins to plateau as it goes on and the learned policy stabilizes. While this effect is not absolute, and our agent is still trending upwards somewhat at the limit of experimentation (74,350 games, a little over 24 hours), it is clearly noticeable. These results suggest that while there is still some policy refinement possible to increase average reward, the *R2*-based agent is starting to converge on what it considers an effective policy, based on relatively stable *Q*-values.

These apparent differences between the results are somewhat mitigated, however, once we restrict our attention to the period beginning at 5,000 games into our testing, where much of the noise and (relatively) atypical behavior from the early games has been overcome. Limiting ourselves to the range of 5000–64,250 games, regression shows reward accumulation in both cases to proceed in upward-trending, linear fashion. This shows that both *R1*- and *R2*-based agents will continue to develop different learned behaviors over time, and that overall learning in game can be expected to be quite slow. These results accord with many observed in the literature, showing the fundamental challenge to learning in such complex domains.

A final set of experiments sought to investigate the effect of a richer action set for our pilot agents. This was driven in part by our observations of highly erratic behaviors, and the hypothesis that this might result from the fact that the agent did not have a complete range of motion, but rather could only turn in one of 8 basic compass directions. We therefore added an additional 8 directions that the agent could turn, allowing it to orient itself every 22.5 degrees. Four RL agents were then run—two with original action sets, and two with the ability to turn in more direction, using each of the rewards *R1* and *R2*—for 80,000 games of learning each. Then, fixing the learned policy for each agent, we ran each for 1050 games to eliminate random outcomes, tracking the number of frames survived per game. This produced one very striking result: using reward-system *R2*, the agent capable of navigating with more precision in its turns lasted over twice as long as any of the others, averaging 6,358 frames per game. Among all of the other

Movement	*Less Turns*		*More Turns*	
Reward	*R1*	*R2*	*R1*	*R2*
Avg. Frames	2091	2451	2481	6358

Table III. Average frames survived over 1,050 games, for all settings of movement and reward.

combinations of reward and action, results were far more comparable, with the *R1* agent that could only turn in 8 directions doing somewhat worse than the others (see Table III). More than rate of reward accumulation, this shows that the *R2* structure can convey significant survival benefits.

6. CONCLUSIONS AND FUTURE WORK

We used reinforcement learning to develop an Xpilot navigational agent. Based on pre-programmed testing, we can conclude that it is indeed possible for an agent to implement a useful and natural navigational policy based on our reduced state space and action set. Although results so far do not show an agent as effective as the pre-programmed bot, our agents did find distinct and interesting strategies, in real time, based on the set rewards. Given the continuing growth in average reward, we also conclude that even longer test runs (on the order of several days compute time) may be required to further craft our agents.

A main conclusion is that complex environments like Xpilot are serious challenges for RL. While agents converged toward value-maximizing policies via *Q*-learning, these were not always the natural and useful navigational policies desired if AI pilots are to be competitive against other players. The main focus of our work now is therefore to find a more suitable reward function. While this is a challenge for any use of RL, our results so far show it to be a promising technique for games like the Xpilot domain, especially since it provides for direct, real- time learning and AI control. We are currently engaged in supplementing RL with evolutionary computing methods, using genetic algorithms to automate the process of developing reward and action parameters, to speed learning, and improve final performance. This approach allows us to do away with the sometimes tedious and *ad hoc* work of designing state, action, and reward abstractions.

Further work will be build up from navigation alone, to allow combat with an enemy bot. After that, we plan to extend to team-based play using multiagent RL, a much more complicated endeavor, relative to its single agent kin, and subject of much ongoing research [9, 10]. Another topic of interest is the use of a "Coach" agent, not directly involved in game-play, but functioning as a learner that can see the position of enemy players and relay strategy to the team. (For one approach, see [11].) Such an agent has the potential to overcome some of the multiagent complexities arising from wholly decentralized approaches.

REFERENCES

1. R. Baumgarten, S. Colton, and M. Morris, "Combining AI methods for learning bots in a real-time strategy game," *International Journal of Computer Games Technology*, vol. 2009, 2009.
2. K. O. Stanley, B. D. Bryant, I. V. Karpov, and R. Miikkulainen, "Real-time evolution of neural networks in the NERO video game," *AAAI-06*, Boston, MA, 2006, pp. 1671–1674.
3. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: MIT Press, 2000.
4. C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, 1992. pp. 279–292.
5. M. G. M. Madden and J. P. Nolan, "Application of AI-based reinforcement learning to robot vehicle control," *Proc.10th Intl. Conf. Applications of AI in Engineering*, Udine, Italy, July 1995.
6. M. Sharma, M. Holmes, J. Santamaria, A. Irani, C. Isbell, and A. Ram, "Transfer learning in real-time strategy games using hybrid CBR/RL," *IJCAI-07*, Hyderabad, India, 2007, pp. 1041–1046.
7. "Xpilot project home-page," http://www.xpilot.org/, last retrieved: 06 February 2010.
8. G. B. Parker and M. Parker, "Evolving parameters for Xpilot combat agents," *Proc. IEEE Symposium on Computational Intelligence and Games*, Honolulu, Hawaii, 2007.
9. Y. Shoham, R. Powers, and T. Grenager, "If multi-agent learning is the answer, what is the question?" *Artificial Intelligence*, vol. 171, no. 7, 2007, pp. 365–377.
10. L. Busoniu, R. Babuska, and B. D. Shutter, "A comprehensive survey of multi-agent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 38, no. 2, 2008, pp. 156–172.
11. P. Riley and M. M. Veloso, "Coach planning with opponent models for distributed execution," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 3, 2006, pp. 293–325.