Adding Smart Opponents to a First-Person Shooter Video Game through Evolutionary Design

C. Adam Overholtzer and Simon D. Levy

Computer Science Department
Washington and Lee University
Lexington, VA 24450
{overholtzerc, levys}@wlu.edu

Abstract

We demonstrate how a first-person shooter (FPS) video game can be made more fun and challenging by replacing the hard-wired behavior of opponents with behaviors evolved via an evolutionary algorithm. Using the open-source FPS game Cube as a platform, we replaced the agents' (opponents) hard-wired behavior with binary "DNA" supporting a much richer variety of agent responses. Survival-of-the-fittest ensured that only those agents whose DNA allowed them to avoid being killed by the human player would continue on to the next "generation" (game). Mutating the DNA of the survivors provided enough variability in behavior to make the agent's actions unpredictable. Our demo will show how this approach produces an increasingly challenging level of play, more fine-tuned to the skills of an individual human player than the traditional approach using pre-programmed levels of difficulty or simply adding more opponents.

Motivation

One of the major commercial applications of AI is in the rapidly expanding computer game industry. Virtually every game sold today has some sort of AI, from the "computer player" in a chess game to the machine-gun-toting enemies in a first-person shooter (FPS). Any virtual being that does not behave in a strictly pre-scripted manner has some sort of AI behind it. Sadly, however, the multi-billion dollar gaming industry has done very little to advance the field of AI. The industry has moved past the day when an AI opponent would not even react to the death of his teammate standing three feet away, but that does not mean these bots, as they are commonly called, have any serious thought processes.

Platform

Cube [1] is an open-source FPS, written in C++ for Linux and Windows, that is based on a very unorthodox engine that emphasizes simple 3D designs to provide a quickly and cleanly rendered environment. The Cube Engine is, to

---

[1] http://cube.sourceforge.net

quote its website, mostly targeted at reaching feature richness through simplicity of structure and brute force, rather than finely tuned complexity. The code for Cube is much shorter and simpler than that for many other FPS games, and, because it is open-source, we have full access to all of the code and thus almost any change is possible .

Approach

We realized that given the simplicity of Cube, developing a straight hard-coded AI that would show real improvement would be a difficult project. Rather than spending lots of time trying to figure out how the agents (bots, monsters, opponents) should behave, we decided that it would be easier and more interesting to add an evolutionary algorithm (Mitchell 1996) to the agent's deterministic behavior. By evolving over time, the agents could learn from experience what the best strategies were and would, for example, decide for themselves how often they should jump over obstacles, instead of running around them.

The idea is that each of an agent's possible decisions are represented by a single value (true, false, or a probability) and all of these values combined determine his behavior. This string of values, which we can call his DNA, is attached to an individual agent. Whenever the agent needs to make a decision, he consults both the usual criteria and the corresponding value in his DNA. (Strings are initially all zero at the start of the first game.) At the end of a game, each agent is given a score based on how well he performed, and five of the ten agents are probabilistically chosen to be "reborn" in the next game, by fitness-proportionate selection. Two copies of each agent are passed on to the next game, but each of these is run through a mutation function that randomly alters a small fraction of the values in the DNA. This way, the agents are changing a little bit each game and the ones that perform the best will live to the next game. Just like biological evolution, our game became a survival-of-the-fittest environment where only the most well-adjusted agents survive and eventually, at least in theory, the agents

become very good at surviving.

Implementation

The main taks in our implementation was to design the DNA sequence and connect it to the existing system. The DNA is a C++ struct consisting mostly of booleans (`caresAboutArmor`, `seeksHealthBoost`), supporting the standard crossover and mutation operators of a genetic algorithms. With this structure in place, it was a relatively simple matter to preface existing hard-wired agent behavior with conditionals like `if (seeksHealthBoost)`, making the agent's behavior contingent on its DNA.

Results

Qualitatively, our project was a success: evolutionary algorithms enabled our agents to move from incompetent to lethal in 50 generations, showing that evolutionary algorithms can be effectively and rather painlessly adapted to 3D first-person shooter games. Furthermore, our simple evolutionary system was extremely fun to play against because the agents grew more and more challenging over time and essentially learned from their mistakes as a real human opponent would. Instead of increasing the difficulty in the cheap way that most FPSes do, either by making the agents stronger or more numerous, our evolutionary algorithm enabled them to legitimately improve their abilities. Because they are computer opponents, they could improve faster than we could, and that is why they could move from morons to masters in relatively few rounds of play.

Quantitatively, we were able to observe the evolution of various "genes" in the agent's DNA over generations. Figure 1 shows the value of `willJump` for each of the five agents that were selected following each of 50 games ("generations"). If an agent is blocked by some obstacle and can jump over, how often should he? Sometime around game 30 they seem to settle on 5, or a probability of 1/3, which is what we originally used when we first wrote the function. But just a few games later, the agents increase the average to around 1/2 and by game 45 or so they seem all seem to agree on jumping 13 times for every 15 possible jumps. This is much higher than we would have expected, indicating that jumping over small obstacles, rather than moving around them, may be a better idea than we thought. Such a result is a good illustration of the benefits of evolutionary algorithms: the agents had evolved a behavior that might be better than what we would have hard-coded.

Future Work

One next step would be to add more capabilities to the agents, such as better navigation or the ability to pick up and use ammo in addition to health and armor. The long-term goal would be to either add some high-level game algorithms in Cube, such as team AI or path-finding, or else implement a similar evolutionary algorithm in a more complex FPS. Our project has shown that an evolutionary algorithm can greatly enhance a first-person shooter, but the real test is whether such a system could push the AI in these games beyond what we have today. If the quality and difficulty of these games can be dramatically improved by a simple off-the-shelf genetic algorithm, then that would be a real achievement.
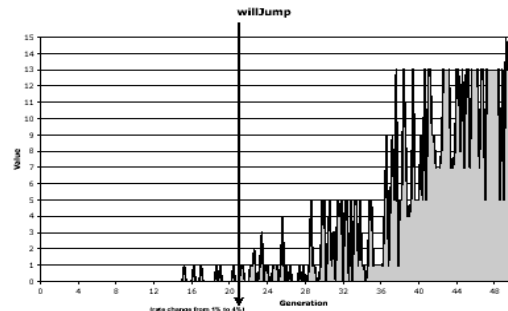

Figure 1. Evolution of a single gene over time

Link to Demonstration

All material needed to play our "EvoCube" FPS on Windows, as well as source code and instructions for further development on Windows and other platforms, can be accessed from:

http://www.cs.wlu.edu/~levy/evocube.zip

Credits

All work described here was done by the first author, as a senior-year independent-study project at Washington and Lee University. The authors thank the developers of the Cube engine for making their game available as a free open-source, project, and the sponsors and managers of the Sourceforge website for hosting Cube and other open-source projects.

References

Mitchell, Melanie M. 1996. *An Introduction to Genetic Algorithms*. Cambridge, Massachusetts: MIT Press.