OpenNERO: a Game Platform for AI Research and Education

Igor V. Karpov[1] and John Sheblak[2] and Risto Miikkulainen[1]
[1] {ikarpov,risto}@cs.utexas.edu
The University of Texas at Austin, 1 University Station C0500, Austin, TX 78721
[2] jsheblak@retrostudios.com
Retro Studios, 1835A Kramer Lane, Suite 100, Austin, TX 78758

Abstract

OpenNERO is an open source game platform designed for game AI research. The software package combines features commonly available in modern game engines (such as 3D graphics, physics simulation, 3D audio rendering, networked play, and a powerful scripting interface) with an easy to use API and tools for defining machine learning tasks, environments, and agents. Flexibility and ease of use of the system are demonstrated by following the process of creating a machine learning game from scratch. The scalability of the platform is tested through the implementation of the existing NERO machine learning game using the new tools.

Introduction

The potential for computer games as a tool for AI research continues to blossom (Bowling et al., 2006). Because games are designed to challenge and entertain human players, they also provide difficult challenges for artificial intelligence. On the other hand, advances in machine learning open the way to creating entirely new genres of video games that use machine learning as an integral part of game play, as demonstrated by Neuro-Evolving Robotic Operatives (NERO). NERO uses real-time neuroevolution of augmenting topologies (rt-NEAT) to allow the player to interactively train control policies for simulated autonomous agents and to test these agents in simulated combat against the opponents' creations (Stanley, Bryant, and Miikkulainen, 2005). While NERO is a successful technology demonstration and enjoys popularity both as a game and a research platform, its development was time consuming and the program is difficult to adapt to new environments, game rules and learning algorithms.

This demonstration introduces OpenNERO, a new general-purpose open source platform that aims to simplify the process of implementing machine learning games and of conducting AI experiments in them. As opposed to recently proposed "glue" systems that can help bind existing games with AI algorithms (Molineaux and Aha, 2005; White, 2006), OpenNERO is a complete software system that allows flexible and easy creation of new games with these AI algorithms in mind. While still under development, the platform has already been successfully used in research and education.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

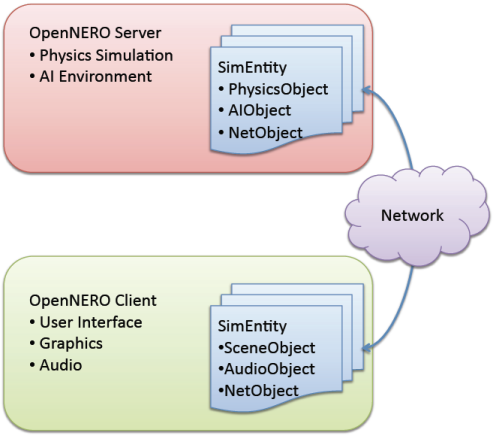

Figure 1: Architecture overview of the OpenNERO engine. The central server simulates the environment and the entities (objects, players or NPCs) in it. The clients are interfaces for human players to use to view the simulation as well as to interact with it.

At the heart of the tool set is the OpenNERO simulation engine that leverages a number of open-source libraries to provide a transparent client-server architecture (Figure 1). OpenNERO combines features commonly available in modern game engines (such as 3D graphics, physics simulation, 3D audio rendering, networked play, graphical user interface, and a powerful scripting interface) with an easy to use API and tools for defining machine learning tasks, environments, and agents. The package also contains an extensible collection of ready-to-use AI algorithms such as heuristic search, value function reinforcement learning, and neuroevolution, as well as tools for empirical evaluation of their performance under different conditions.

Building a Machine Learning Game

A good way to demonstrate the flexibility and ease of use of the OpenNERO platform is to create a 3D environment, a

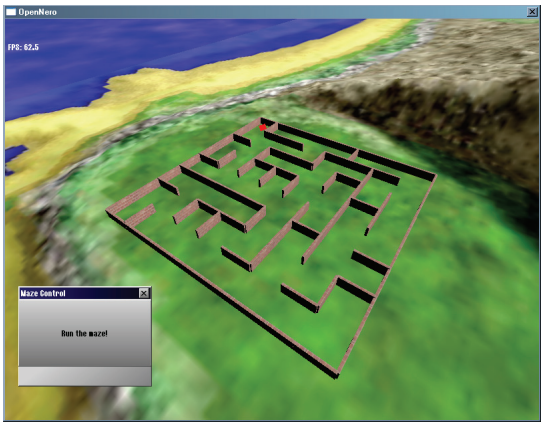

Figure 2: Screen-shot of the maze environment. The maze is randomly generated using a scripted Kruskal's algorithm. Such an environment can be used to demonstrate heuristic search methods as well as machine learning methods such as neuroevolution or reinforcement learning, as well as allowing the human players to try the task for themselves.

simple game that involves a reinforcement learning task, and to solve this task by using an existing reinforcement learning agent. In order to do so, the user defines a randomly generated 2D maze with a start state and a goal state, defines the task (the rules and the reward structure) of the game, and defines two characters (player-controlled and non-player-controlled) that will play this game. Each of the three components of this creation process - the environment, the task and the agent - is described in more detail below.

- *Environment* - OpenNERO provides two tools, Importer and Builder, that simplify the creation of virtual environments. The Importer tool allows the user to load 3D models in a variety of formats and to set OpenNERO specific properties such as physical attributes, animations, and sounds. The Builder tool allows the user to place different objects in the environment either manually or programmatically and to preview the resulting world. It also allows the user to adjust global constants that affect the underlying physics simulation. For the maze environment, the environment consists of an island terrain and a script that places the walls of the maze generated using Kruskal's algorithm.
- *Task* - The user can define tasks in the game as a reinforcement learning problem by defining the environment logic that takes in actions to produce sensors and rewards that are then consumed by the players or non-player characters. The sensors can query the simulation using parametrisation of a number of predefined queries such as a virtual camera, a virtual microphone, virtual range sensors, object counters and filters. Actions are currently represented as vectors with variable number of dimensions; individual components of an action vector have a minimum and a maximum value and can be either discrete or continuous. The reward is either a single continuous number or a set of several rewards along multiple objectives with their associated "importance" weights. For the maze game, the agent controls its angular and linear velocity, receives a constant penalty for performing non-terminal actions and a large positive reward for reaching the goal.
- *Agent* - The user defines the *start*, *act* and *end* methods of an agent in script. Here, the user has all the algorithms implemented in OpenNERO at his or her disposal. Example controllers for the maze environment include manual control, A* search, Q-learning and neuroevolution.

In OpenNERO, creating environments, tasks and agents can be done with only a small amount of Python code and without recompiling any platform-dependent code. The end result is a simple but full-featured 3D video game that can be used as a visual aid for comparing different AI algorithms (Figure 2).

Scaling OpenNERO to NERO

In order to test the scalability of OpenNERO, the existing real-time neuroevolution game NERO is being reimplemented using the new tools. NERO is a computationally intensive game because it requires the simultaneous calculation of ego-centric sensors and evaluation of control policies for a population of 50 or more autonomous simulated agents every decision time step (usually on the order of 0.1 sec). Currently, the NERO implementation matches the capacity of an average consumer machine at population sizes under a hundred agents. In the future, the OpenNERO platform will be further extended to allow decision making to be distributed over a network of computers, which should reduce the load on the central simulation server and make it possible to build more complex games.

Acknowledgments

This research was supported in part by NSF under grant IIS-0757479, Texas Higher Education Coordinating Board under grant 003658-0036-2007, the College of Natural Sciences at University of Texas under the FRI program, and the IC^2 Institute of the University of Texas. OpenNERO uses several software packages made available by their respective authors via licenses approved by the Open Source Initiative.

References

Bowling, M.; Fürnkranz, J.; Graepel, T.; and Musick, R. 2006. Machine learning and games. *Machine Learning* 63:211–215.

Molineaux, M., and Aha, D. W. 2005. TIELT: A testbed for gaming environments. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (Intelligent Systems Demonstrations)*. AAAI Press.

Stanley, K. O.; Bryant, B. D.; and Miikkulainen, R. 2005. Evolving neural network agents in the NERO video game. In *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games(CIG'05)*. IEEE.

White, A. 2006. A standard system for benchmarking in reinforcement learning. Master's thesis, University of Alberta.