Machine Learning in a Dynamic Game Environment

Project Proposal
Joe Strout
November 7, 2012

1 Introduction

Modern video games place the player in a virtual world, interacting with objects and other characters. In many games, virtual objects react in very realistic ways, due to detailed physics simulation. The non-player characters (NPCs), however, often behave in unrealistic ways, due to the inherent complexity of simulating human behavior. These artificial intelligence (AI) failures can often be quite jarring, disrupting the player's suspension of disbelief.

AI failures come in many different forms, ranging from inadequate path-finding, to poor sensory simulation, to insufficient situational awareness. For this project, my focus will be one one particular category of problem: failure to learn from experience. This becomes apparent when the player discovers (or the NPC stumbles upon) some pattern that causes an NPC to fail, and that pattern works again and again, with the NPC never learning from its mistake. For example, a commonly seen failure is an NPC that has gotten "caught" on some obstruction in the terrain, and runs in place indefinitely. A learning agent would realize that it's making no progress with that approach, and try something else.

In most current games, designers attempt to limit such failures by doing extensive (and time-consuming) analysis of each level (simulated world) during development, and baking this information into the level data for quick reference by the AI engine. However, a new class of games, exemplified by Minecraft, allow players great freedom in shaping both the terrain and the behavior of the world. For example, players can create or clear obstacles and paths at whim, and even set up complex logical triggers causing doors to open and close, traps to appear, etc. In such an open-ended, player-driven world, extensive pre-game level analysis is impossible. If an NPC is to maintain a believable illusion of intelligence under such circumstances, it must be able to learn from experience, so that it will not get trapped in an infinite loop of failure.

2 Experimental Design

A simple 2D world will be defined, consisting of at least the following elements: walls (both visible and invisible), pits, doors, keys, buttons, moveable stones, and gems. A game will be run with each of several different AIs (described below), as well as several human players for comparison. Each game will be divided into a number (probably 5 to 10) of levels, each defined by an initial layout of elements, plus level-specific logic that causes elements to appear or disappear in response to events. For example, a passageway might be opened or closed when a button is activated, or simply when the player agent steps on a certain grid cell.

On each turn, the player will have a limited number of available actions: move in any direction, pick up or drop an object, or wait. Each level will be run for some fixed number of turns (perhaps 1000), after which the player is immediately advanced to the next level. The object of the game is to maximize a score, where score points are obtained by collecting gems, and possibly lost through certain hazards.

Levels will be designed to present challenges of the sort that should be easy for a human, but that would likely stump a nonadaptive AI. A simple example would be an invisible wall blocking an obvious path to a gem. To the AI's sensors, this cell is indistinguishable from any open cell, but the game engine will not allow the agent to enter it, simulating the case in modern games where an NPC gets caught on some obstruction. As a more complex example, imagine a passage leading to a gem has doors on both ends, one open and one closed; a button in the middle toggles them both. Even though there is never a clear path all the way through, a smart player will quickly learn to enter, hit the button, and continue on through.

Levels will also be designed to provide an infinite supply of gems, by simply including some logic to spawn a new gem when a gem is taken, requiring the player to repeat some (possibly complex) action to collect more. This should ensure that the more quickly a player learns to deal with each level's challenge, the that player's score will be.

3 AI Design & Implementation

I would like to try several different designs for the AI, including a "control" AI that has no learning at all, but instead uses a simple path-finding algorithm to approach the nearest gem. On the surface, this appears to be a classic reinforcement learning (RL) problem. However, my intuition is that an RL algorithm will be too slow to master each level and collect many gems within the allotted turns. I may try that as one of the AIs anyway, but I also intend to try a different design, where the goal of the AI is to predict the state of each grid cell at some point in the future. Those predictions will then be used by a path-finding algorithm to make a plan, but one that takes into account expectations about the world learned through experience. For that learning problem, I might keep a small neural network per grid cell, whose inputs are things like the player position, the state of all buttons on the level, the turn number (modulo 2 through 10), and so on. As a different AI, I might approach the same learning problem with a decision tree.

I plan to implement the game simulation and AI in Real Studio, a cross-platform, fully-compiled development environment. This will require recoding the algorithms learned in this class, but has the advantage of producing code I can use in future (work-related) projects I have in mind. I have already prototyped the game simulation, and have existing code for path-finding and decision trees that I can reuse. A screen shot of this prototype is provided in Figure 1.

Figure 1: Screen shot of game engine prototype, illustrating player avatar and a variety of environment elements.

4 Milestones

Nov. 12: Finish the game simulation, including level layouts and logic. The game should be playable by a human at this point.

Nov. 19: Nonadaptive path-finding AI will be working at this point. Work on learning AIs should be underway.

Nov. 26: Finish both neural network and decision tree implementations. Tweak parameters and input encoding to maximize the effectiveness of each AI.

Dec. 3: Complete experiments, running final versions of all AIs, as well as three human subjects (my spouse and two sons).

Dec. 10: Finish written report and presentation slides.