Machine Learning in a Dynamic Game Environment

Joe Strout CS545

Luminary Apps

My name is Joe Strout, with software engineering firm Luminary Apps.
My project explores machine learning in a dynamic game environment.

Talk Outline

- Problem Overview
- Experimental Setup
- AI Architectures
- Results
- Conclusion

I'll begin by defining the problem I'm trying to solve.
I'll then describe the experimental setup — I'm comparing five different versions of the AI to three human subjects.
I'll describe the architecture of the five AIs, and then present some representative results from the experiments.
Finally, I'll summarize the results and try to provide some useful conclusions.

The Problem

- Poor AI disrupts suspension of disbelief
- Game environments increasingly malleable
- Traditional tricks can’t adapt

> “...The worst A.I. I've seen in awhile. It definitely ruined the game for me, took me out of the experience and just made it not fun to play.”
> *- internet comment about Crysis 2*

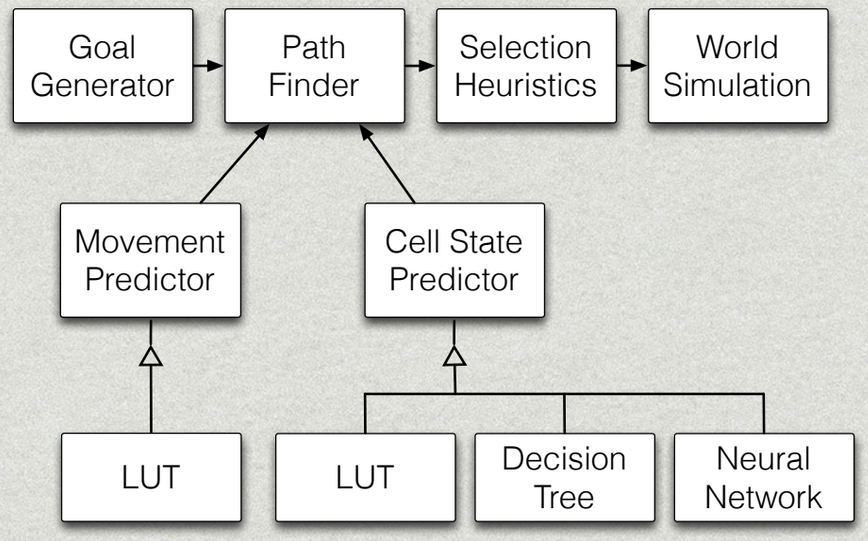
First, the problem.
Part of the “suspension of disbelief” when playing a game is to suppose that the non-player characters sharing the world with you are people, with real minds and motives. When an AI agent does something obviously stupid, like getting stuck on some invisible obstacle, this illusion is broken, and the player’s immersion in the game is disrupted.
On top of that, there is a trend toward more complex and malleable environments. Minecraft is the most famous example: the whole world can be edited by players, even setting up triggers and actions that make the world react to the movement of agents within it.
Traditional AI techniques can’t adapt to such real-time changes; machine learning is required.

Experimental Setup

- 10 x 10 world
- Multiple objectives
- Multiple obstacles
- Dynamic environment
- 8 levels, 150 turns/level
- 5 AIs, 3 humans

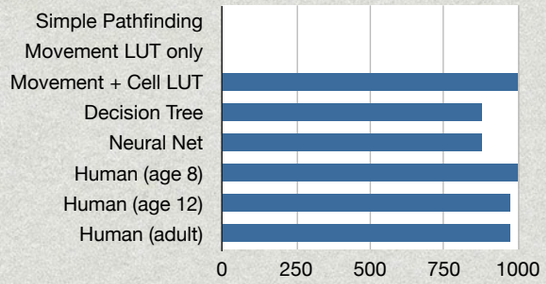
To explore this issue, I created a simple world simulation consisting of a 10x10 grid, which is navigated by a human or AI agent. In this grid are multiple objectives — gems, which can be collected for points, as well as rocks and keys that can be carried around. There are also multiple obstacles to be navigated around.
The environment is dynamic: logic specific to each level causes the map to change based on the state of buttons, the position of the agent, or the clock. Walls can appear or disappear, doors can be opened, bridges can move or disappear, and so on.
Each experimental run is divided into eight levels, with 150 turns per level.
All code was written in Real Studio, based upon code from this class as well as other sources.

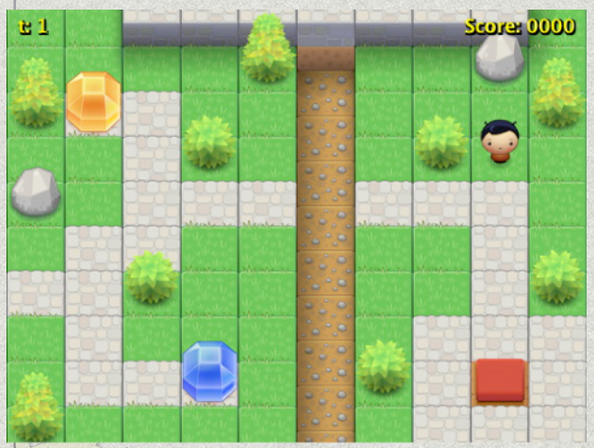
AI Architecture



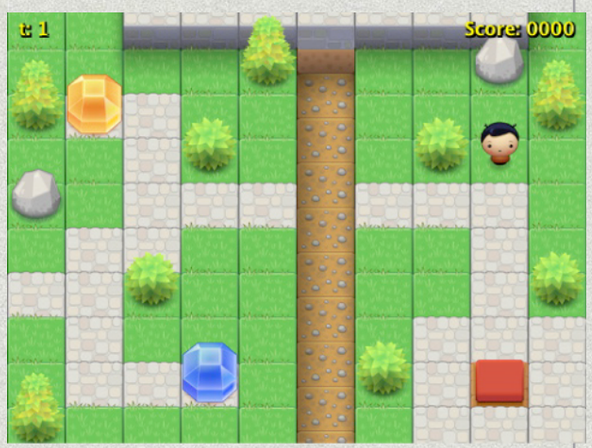
Next, let me give a block-level overview of the architecture of the AIs.
The most basic AI is a straightforward pipeline: a goal generator that finds the locations of key points in the map (gems, buttons, keys, and so on); a path finder, which uses a standard A* algorithm to plot a course to each goal. When more than one goal is reachable, some simple heuristics are used to select the best one, and the resulting action — the first step on the chosen path — is fed into the world simulation. The whole cycle then repeats for the next turn.
Many commercial games stop here, so I included this as my most basic AI.
The next step beyond that to predict whether an attempted movement will actually succeed, and learn where it does not. That's the movement predictor module. Finally, the real meat of the learning part is something to predict the state of every cell the board in the future. This includes obstacles, gems, doors opening or closing, and so on.
For the movement predictor, I first tried a simple look-up table approach. This worked so well that I couldn't bring myself to implement any more complex algorithms. But for the cell state predictor, which is more the focus of the project, I implemented three different algorithms: a look-up table, a decision tree, and a neural network, trained with SCG back propagation. (20 hidden units, 10 training iterations per turn)

Level 4





(Cell State) LUT

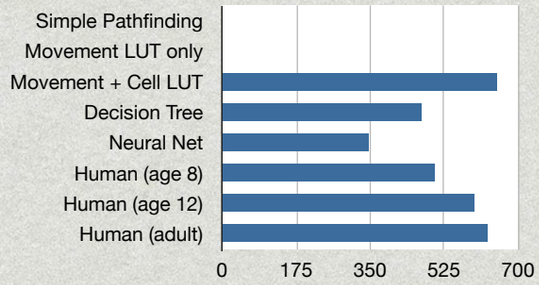


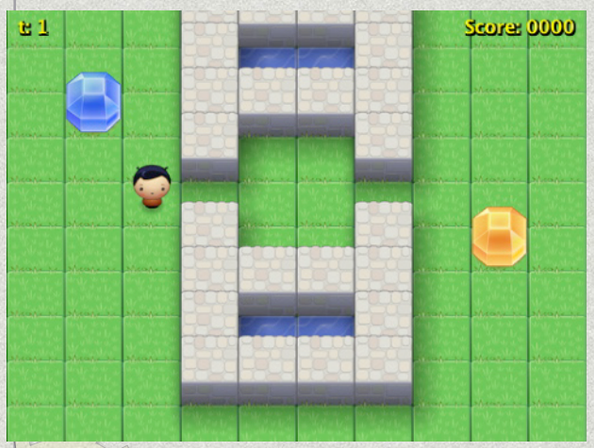
Decision Tree

On this level, the movement plus cell lookup table performed best.
The decision tree and neural network took longer to find the solution.

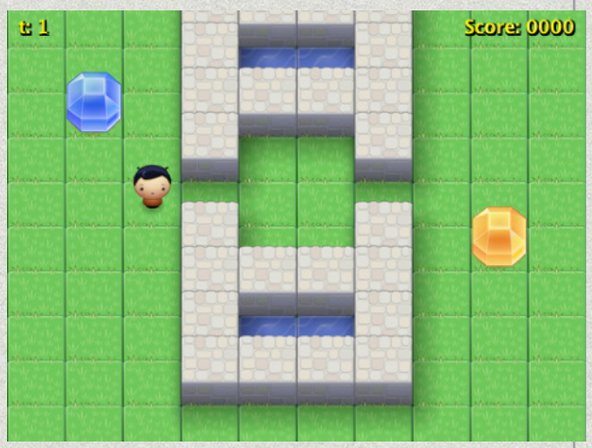
The non-learning AI, and AI with only obstacle lookup table, failed completely. They are unable to foresee the benefit of leaving a rock on the button.

Level 5





(Cell State) LUT



Neural Net

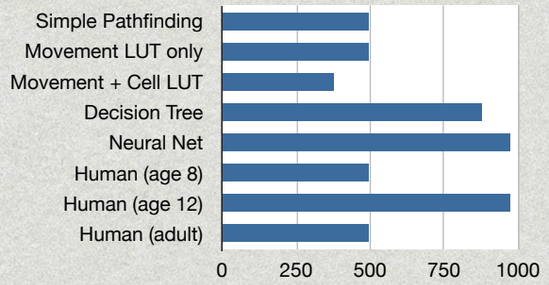
This level, described by one of the human subjects as “mean,” blocks off an obvious passage every time you attempt to use it. But this happens only in one direction.

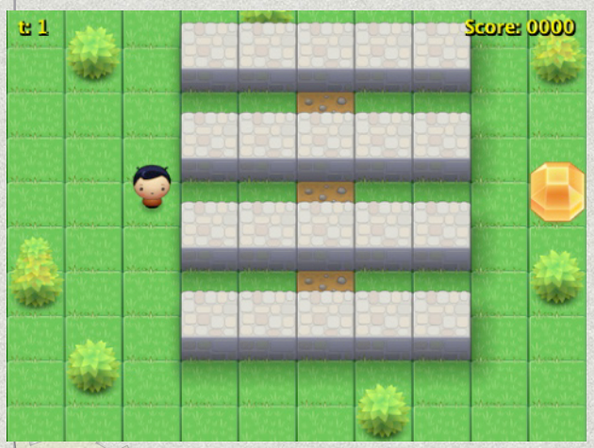
The cell state LUT quickly found this solution, and achieved a very high score, higher even than human subjects.

The neural network took much longer to accept that it couldn’t take the short way through to the right, and received the lowest score of the full AIs. (Again the simpler AIs failed completely, simply cycling through stepping into and out of the center passage.)

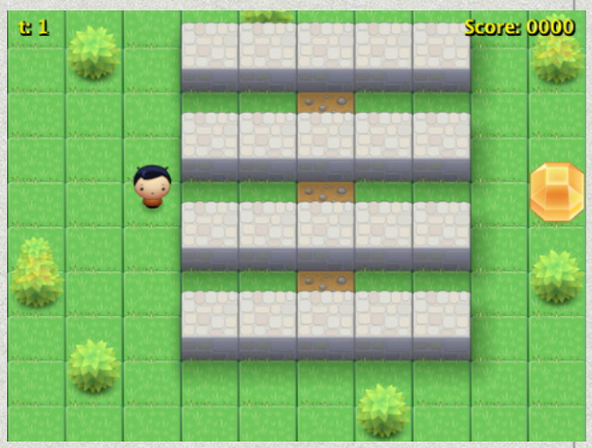
Interesting note: the decision tree (not shown here) quickly learned to avoid the shortcut to the right, but overgeneralized, and also took the long way back to the left. The 8-year-old human subject did exactly the same thing.

Level 7





(Cell State) LUT



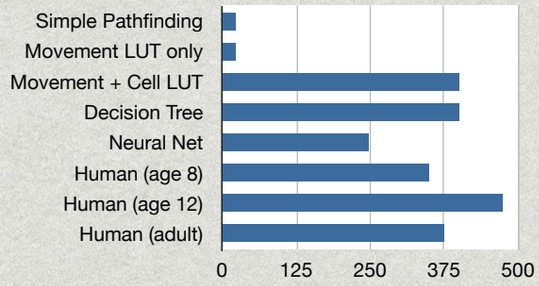
Neural Net

Here you can see that the cell state LUT gets completely “flustered” by the periodically appearing and disappearing walls.

The neural network, on the other hand, learned the pattern, and found the strategy of simply going down the center, waiting a few turns as needed. This is the same strategy used by the 12-year-old human subject.

This is the only level on which the neural network beat the decision tree, though not by much.

Level 8



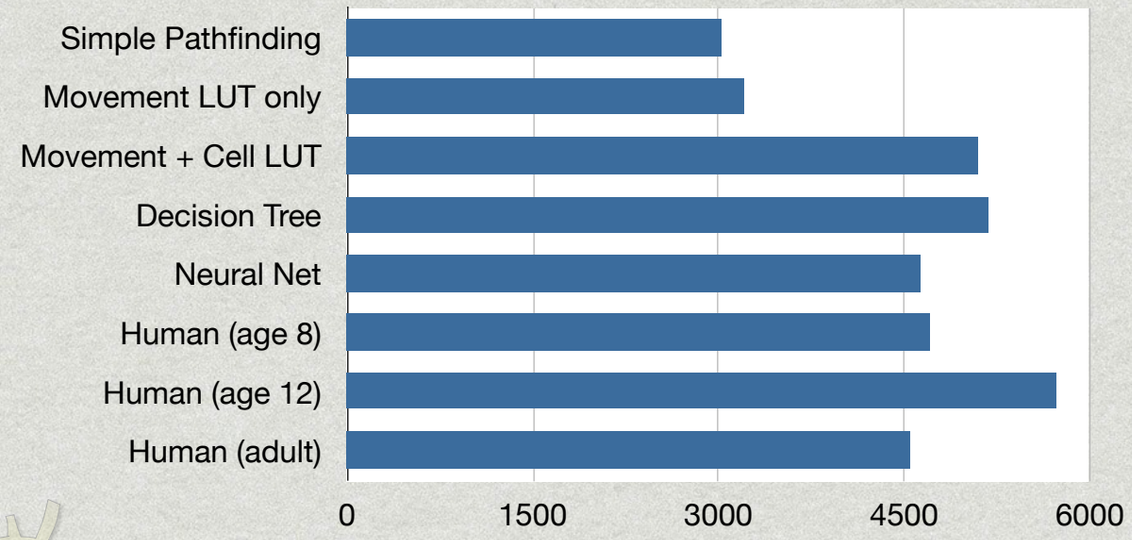
Neural Net

Decision Tree

This final level was meant to employ all the tricks I could think of in one place, testing overall performance in a realistic dynamic environment. The bridge over the ditch moves every 2 turns, and the bridge over the water is controlled by a button. In addition, there are keys and rocks to be collected.

The neural network in this case took longer to learn that it was the button being down, rather than stepping off of it in some particular way, that caused the water bridge to appear.

Total Score/Conclusions



Conclusions:
1. There's a sizeable gap between a nonlearning AI and even the poorest of human players.
2. A simple lookup table to track impassable terrain helps avoid the most obvious failures, but is quickly tripped up by a dynamic environment.
3. In a dynamic environment, any learning algorithm tested produced dramatically better performance.
4. Of the ones tested, the decision tree produced the best score overall, and never fell into obvious traps.