Teaching Reinforcement Learning with Mario: An Argument and Case Study

Matthew E. Taylor
Lafayette College, Easton, PA
taylorm@cs.lafayette.edu

Abstract

Integrating games into the computer science curriculum has been gaining acceptance in recent years, particularly when used to improve student engagement in introductory courses. This paper argues that games can also be useful in upper level courses, such as general artificial intelligence and machine learning. We provide a case study of using a Mario game in a machine learning class to provide one successful data point where both content-specific and general learning outcomes were successfully achieved.

Introduction

When considering possible assignments in undergraduate courses, it can be critical to select topics and exercises to properly engage students. Computer games have been used successfully in both introductory computer science courses (Parberry, Roden, and Kazemzadeh 2005; Bayliss 2007) and general artificial intelligence classes (DeNero and Klein 2010; Wong, Zink, and Koenig 2010) to build excitement and enthusiasm. However, there has not yet been widescale adoption of games. This paper makes the argument, focused on the AI community, that games can be successfully incorporated into the undergraduate curriculum while meeting pedagogical requirements (discussed on the following page), and without requiring a large investment in infrastructure. We believe that games can also be successfully introduced into graduate courses, but focus on undergraduate studies because 1) the case study in question was conducted in an undergraduate class and 2) the excitement that games provide is likely less critical to graduate students (who are presumably already at or near the top of Bloom's Taxonomy (Bloom 1956) and more self-motivated).

In addition to incorporating games into the computer science curriculum, a growing numbers of schools are developing game development programs (e.g., The NYU Game Center[1] and USC's GamePipe project[2]), which are popular both among a diverse set of students, and with industry recruiters (Kessler, van Langeveld, and Altizer 2009). Such programs demonstrate that there is significant demand for such topics among the students. Furthermore, the link between machine learning and mainstream games is becoming stronger in recent years. For instance, machine learning has become not only a component of the game, but in some cases is the *focus* of the game. For instance, the creature in the adventure game Black & White is trained by the user to accomplish specific tasks — the creature learns via a decision tree algorithm (Barnes and Hutchens 2002). In the game of NERO, the user's goal is to train a team of agents, controlled by a population of neural networks, to complete military tasks (Stanley et al. 2006). Given the integration of AI into the current generation of games, it makes all the more sense to leverage games in our courses.

This paper argues that computer games are a particularly appropriate platform for artificial intelligence classes and provides a case study with a reinforcement-learning focus. In the case study, we show how students were able to scaffold information through the game, achieving both course-specific goals (i.e., understand and implement multiple learning algorithms), as well as to improving process goals (i.e., be able to empirically analyze and compare multiple algorithmic approaches). An ideal case study would examine two classes covering the same material: one using games to reinforce concepts and the other using more traditional assignments. Unfortunately, this comparison is not available. However, we will provide our reasoning for the project design decisions, uantitative and qualitative evidence of success, and lessons learned from the experience. It is our hope that this paper will help encourage others to integrate games into machine learning classes, as well as other AI classes in general. We believe there are 1) significant gains to be had by capturing student's enthusiasm, 2) sound pedagogical arguments for incorporating games into our classes, and 3) that the additional overhead of interfacing with such games is manageable.

Context

An Introduction to Machine Learning was offered for the first time at Lafayette College in the fall term of 2010.[3] The course had an enrollment of nine students (typical for upper

[1] http://gamecenter.nyu.edu/

[2] http://gamepipe.usc.edu/

[3] The class webpage, including assignments, slides, and readings, may be found at http://www.cs.lafayette.edu/~taylorm/cs414/index.html.

level electives), of whom seven were juniors and two were seniors. The majority were B.S. students in computer science and the remainder were B.S. students in mathematics or electrical and computer engineering. Roughly half of the students had taken the *Artificial Intelligence* class as an elective in the fall of 2009.

There were no explicit prerequisites for the class, although all students were expected to be able to write "non-trivial" programs in Java, Python, C, or C++. Course objectives included:

- Discuss the goals and current limitations of machine learning algorithms,
- Provide familiarity with a variety of fundamental algorithms, and
- Present contemporary machine learning conference or workshop papers to the class.

The course was split into three sections. The first provided background on general classification and regression techniques, using Mitchell's textbook (Mitchell 1997). The second focused on reinforcement learning, covering chapters 2–9 of Sutton and Barto's textbook (Sutton and Barto 1998), building upon the first section of the class that discussed abstraction and function approximation. The third discussed a number of more recent topics in both classification and reinforcement learning. The students' grades were based on the successful completion of three projects corresponding to the three sections of the class. This paper focuses on the course's second project, reinforcement learning in the Mario domain.

Games for Teaching

As discussed in the introduction, studies show that games can serve as a motivational tool for computer science students. Fortunately, there are also many existing APIs and platforms which allow students and researchers to interface with commercial games. Additionally, there have been multiple games developed specifically for research. In the former category, for instance, AIIDE has released an API as part of their annual Starcraft tournament[4] providing an interface to the popular 1998 real-time strategy game by Blizzard Entertainment, others (Amato and Shani 2010) have developed and released an API[5] for the turn-based game of Civilization IV, and Wargus[6] was developed to provides an interface to Warcraft 2, a classic real-time strategy game from 1995. In the latter category, there are platforms developed specifically for competitions and research, such as The Open Racing Car Simulator[7] (TORCS), which focuses on real-time robotic control, and Generalized Mario[8], which is the focus of the current paper.

[4] http://eis.ucsc.edu/StarCraftAICompetition

[5] http://research.microsoft.com/en-us/downloads/2aae21f8-3d2a-4926-9208-167df090b0cf/

[6] http://wargus.sourceforge.net/

[7] www.berniw.org/tbr/index.php

[8] 2009.rl-competition.org/mario.php

Our decision to use Mario was motivated by both pedagogical and technical reasons. First, consider the pedagogical requirements of using a game in a teaching setting. For instance Repenning, Webb, and Ioannidou, said that for

> "... systemic impact, a computational thinking tool... must fulfill all these conditions:
>
> 1. Has low threshold: a student can produce a working game quickly.
> 2. Has high ceiling: a student can make a real game that is playable and exhibits sophisticated behavior, e.g., complex AI.
> 3. Scaffolds flow: the curriculum provides stepping stones with managed skills and challenges to accompany the tool.
> 4. Enables transfer: tool + curriculum must work for both game design and subsequent computational science applications as well as support transfer between them.
> 5. Supports equity: game design activities should be accessible and motivational across gender and ethnicity boundaries.
> 6. Systemic and sustainable: the combination of the tool and curriculum can be used by all teachers to teach all students (e.g. support teacher training, standards alignment etc)." (Repenning, Webb, and Ioannidou 2010)

Although their research focused on creating games in K-12 settings with multiple instructors, we believe that points 1–5 are equally applicable when using games as a teaching tool in college courses. We next discuss how the Mario domain fits the above criteria.

The Mario platform was developed as part of the RL-Competition[9], a semi-annual international competition designed to compare different competitors' learning algorithms. One of the key design decisions, which is equally critical in an educational setting, is that all learning tasks be easy to install and understand. If a competitor in the competition cannot easily generate a simple working agent, s/he is less likely to participate in the competition (criterion #1). The Mario domain is quite complex: it has a very large state space (as discussed in the next section), there is a large action space, there are multiple levels of strategy available to the agent, etc. A high-performing agent must be fairly sophisticated and there are many possible strategies to improve overall performance (criterion #2). As discussed later in this paper, students were required to implement multiple algorithms in the Mario domain, building upon their previous implementations. Further, many students used the Mario domain for a second class project. Both of these points speak to criteria #3-#4. In fact, we believe that Mario could also be used for teaching other machine learning techniques, or even general AI techniques, which speaks to the flexibility of games as a teaching platform. One potential shortcoming of using games is that, stereotypically, they have more appeal to males than females. In the case of this course, the

[9] www.rl-competition.org

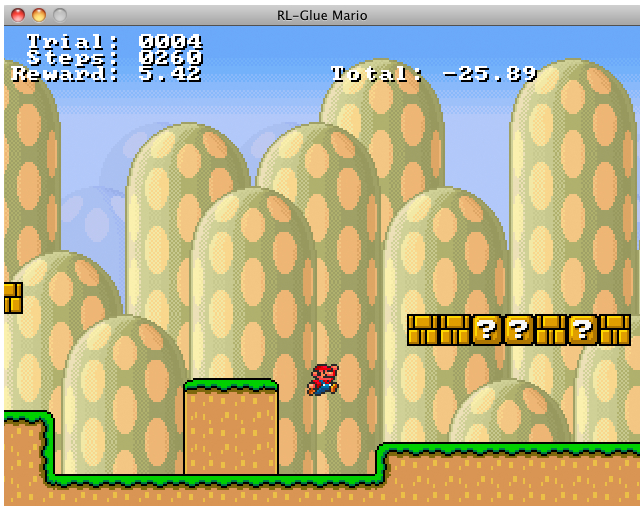

Figure 1: A screenshot of the visualizer distributed with the Generalized Mario domain

question was moot as the class was entirely male. Further, not all students in the class had played the original Mario game, but the concepts were immediately recognizable from contemporary games (criterion #5).

The technological reasons for selecting Mario as a platform were fourfold. First, we wanted students to be able to run programs on the computer science lab machines, which run Ubuntu. By ensuring Linux compatibility, students were able to gather data on the cluster, significantly enhancing their ability to make statistically valid claims (relative to having to run experiments on personal computers). Second, we did not want to force students to use a particular language. The RL-Glue platform supports Java, C, C++, Lisp, Matlab, and Python, allowing the students to select the language that they were most comfortable with. Third, the Mario domain (as well as many other RL-Glue domains) has a robust visualizer (see Figure 1), which assists the students in debugging and understanding their agent's policy. Fourth, the Mario domain was also able to run "headless" (i.e., without the visualizer) so that experiments could be run without waiting for graphical calculations.

Taken as a whole, we believe that the Mario domain fit the needs of the course, but (as emphasized above) there are many options available to instructors who wish to utilize games to teach machine learning concepts.

The Mario Project

This section will briefly discuss the assignment in the Mario domain. Full details, as well as materials to easily integrate the project into other courses, can be found elsewhere (**?**). Before detailing the goals of the Mario project, we first introduce key concepts and terms in reinforcement learning (RL) problems.

Reinforcement Learning Background

RL problems are typically framed as *Markov decision processes* (MDPs) defined by the 4-tuple $\{S, A, T, R\}$. An agent perceives the current *state* of the world $s \in S$ (possibly with noise). Tasks are often episodic: the agent executes actions in the environment until it reaches a terminal or goal state (e.g., the agent "dies" or completes a level), at which point the agent is returned to a starting state. The set A describes the *actions* available to the agent, although not every action may be possible in every state. The *transition function*, $T : S \times A \mapsto S$, takes a state and an action as input and returns the state of the environment after the action is performed. The agent's goal is to maximize its reward, a scalar value defined by the *reward function*, $R : S \mapsto \mathbb{R}$. In games, T and R are typically controlled by the game environment and not explicitly known by the agent (although they may be easy for a human to intuit).

A learner chooses which action to take in a state via a policy, $\pi : S \mapsto A$. π is modified by the learner over time to improve performance, defined as the (discounted) expected total reward. Instead of learning π directly, many RL algorithms instead use a *temporal difference* method to approximate the action-value function, $Q : S \times A \mapsto \mathbb{R}$. Q maps state-action pairs to the expected real-valued return (Sutton and Barto 1998). If the agent has learned the optimal action-value function, it can select the optimal action from any state by executing the action with the highest action-value. In tasks with small, discrete state spaces, Q and π can be fully represented in a table. As the state space grows, using a table becomes impractical, or impossible if the state space is continuous. Agents in such tasks typically factor the state using *state variables* (or *features*), so that $s = \langle x_1, x_2, \ldots, x_n \rangle$. In such cases, RL methods use *function approximators*, such as discretization, artificial neural networks, or tile coding, where parameterized functions representing π or Q are tuned via supervised learning methods. The parameterization and bias of the function approximator define the state space abstraction, allowing observed data to update a region of state-action values rather than a single state/action value.

Reinforcement Learning and Mario

This section provides an introduction to the *Generalized Mario* domain, first introduced as part of the 2009 RL-Competition (Whiteson, Tanner, and White 2010), based on the 1983 Mario Bros. side-scroller by Nintendo Co., Ltd. On every time step, the Mario agent must select an action in a discrete three-dimensional action space: dimension 1 is {-1, 0, 1} and corresponds to "moving left," "not moving," or "moving right"; dimension 2 is {0,1} and corresponds to "not jumping" and "jumping"; and dimension 3 is {0,1}, corresponding to "running" and "walking." This three dimensional action space can also be thought of as an action space with 12 distinct actions. The agent receives state information as a 21×16 array of tiles, representing the state space corresponding to a (human's) view of the game world. Each tile contains information about whether Mario can travel through the tile, if Mario can walk on top of the tile, etc.

The agent's goal is to learn a policy that can receive large amounts of reward. In this project, we focused on learning an action-value function, Q, which accurately predicts the long term reward for an action in a given state. Mario receives a small negative reward on every timestep, a negative reward for dying, a positive reward for collecting coins or power-ups, and a large positive reward for finishing the level (i.e., traveling from the start state, at the far left of the level, to the goal line, at the far right of the level).

Topics Covered

The Mario project, the second of three projects in the course, was designed to reinforce multiple RL concepts and topics. The most fundamental is that of state representation. On every time step, the world is represented by a 352-element array. There are 336 elements corresponding to tiles in the world, where each tile is one of 14 possible types (i.e., can the agent walk through this tile, can the agent stand on this tile, etc.). The additional array elements contain information about any monsters that are present. There are a varying number of monsters present at any given time and there are nine different types of monsters, significantly adding to the state complexity. Rather than reasoning over a huge number of possible states, some type of function approximation must be used.

Students were required to submit proposed state representations to the instructor before implementation so that the representation could be refined. Most students discretized the state space into 10,000-100,000 possible abstracted states. Fewer states may allow an agent to learn faster, but such states may be less expressive, leading to lower final performance.[10] The student's state representation is likely to have a large impact on an agent's learning ability, but different sections of the project were designed to be evaluated in a relative manner. Even a poor choice of state space will allow the student to show improved or decreased performance when implementing different algorithms.

In the project, students implemented Sarsa, a popular temporal difference learning method (Sutton and Barto chapter 6) and had the option to implement two additional temporal difference learning methods: Monte Carlo (chapter 5) and Q-Learning (chapter 6). Students were expected to compare and contrast the different methods, as well as hypothesize why different methods outperformed others. We expected that getting Sarsa to work initially would be the most difficult step in the project.

After implementing these learning methods, students extended the basic implementation in two different ways. First, they tried using eligibility traces (chapter 7), which are a way of generalizing experience over time. Second, they could use *transfer learning*, where an agent trained on one level of Mario, and then attempted to learn a second (different, but related) level.

[10] Additionally, the Markov Property is increasingly violated as the state space uses a smaller discretization, which results in environment states with different optimal actions getting mapped to the same internal state representation.

Project Analysis

The RL-Competition was designed so that students and researchers would be able to easily install the programs and be able to quickly begin developing. Students reported that this goal was achieved — the installation was relatively fast and that the code provided was easy to understand.

In some RL tasks, the state and state variables are hard to mentally model. In video games, the entire state can be represented visually and students are more comfortable with the agent's decision making process. Even though not all students had played the Mario Bros. game, their discussion during class and write-up of their state discretization showed that the video game environment allowed them to easily conceptualize the agent's state.

Once a state space was selected, students next task was to implement Sarsa. Figure 2 shows one student's Sarsa learning curve, demonstrating successful improvement. Students benchmarked their Sarsa learning algorithm against both a random policy and against an example policy included in the RL-Competition distribution. While eight of the nine students successfully outperformed the random agent (and therefore had a working implementation of Sarsa), none outperformed the hand coded policy included in the RL-competition downloadable package.

Six students chose to implement Q-Learning in addition to Sarsa, and two chose to implement Monte Carlo (as discussed in the example projects submission, students were allowed flexibility as to which sections of the assignment they could choose to complete). Of the students who implemented Q-Learning or Monte Carlo, all except one completed a good or adequate analysis. An assessment of "adequate" required that the students perform a statistical comparison and propose a hypothesis to explain the algorithms' relative performance.

Eight students implemented eligibility traces by programming the Sarsa(λ) algorithm. Five of the eight students did a good job in their analysis, making reasonable arguments as to why the eligibility traces did or did not improve performance. Two of the students provided a partial analysis, and one student failed to analyze his results completely.

All nine of the students studied the effects of transfer learning in the Mario domain by comparing their agent's performance on level 1 when learning without prior knowledge with their agent's performance on level 1 after previously learning on level 0. The majority of students found that transfer did not improve learning, as the two levels were significantly different. Of the nine students, six provided good analyses of the transfer learning behavior, as well as possible explanations of their findings, and an additional student provided partial analyses.

Analysis of Outcomes

At the end of the 3.5 week project, all nine students were able to analyze their agents' performance by generating learning curves, the minimal requirement for the project. Eight of the nine students were able to demonstrate that their Sarsa agents improved performance over time, indicating a successful implementation of an RL algorithm in a complex

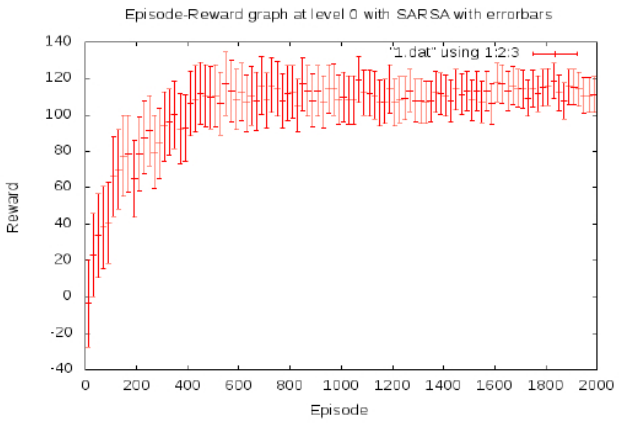

Figure 2: This figure shows an example learning curve from a student's report. A Sarsa agent successfully learns on the "level 0" task, where the x-axis shows the number of episodes, and the y-axis shows the reward (i.e., average performance per episode). Error bars show the standard deviation over 30 trials.

domain. Additionally, a majority of the students were able to successfully implement additional RL algorithms and compare their performances, which contributed to the first and second desired outcomes for the course as a whole. The seven students who implemented transfer learning, and provided complete or partial analyses of the results, were able to do so by reading and understanding the class discussions related to transfer learning papers, contributing to the third desired outcome for the course.

One challenge throughout the class was to shift students' focus from "getting the program completed" to "understanding the algorithm's performance." For many of these undergraduate students, this was their first exposure to a research-oriented course. Rather than simply learning new material and applying it in practice, they were expected to conduct experiments analyzing their programs and ideas. While not an explicit objective of the class, the number of students who were able to successfully analyze their programs did increase between the Mario project and their final project. However, since the second and third projects were substantially different, we cannot claim that this perceived improvement was directly caused by the Mario project.

Finally, the students' end-of-semester course assessments indicated that the project was indeed useful, as they included comments like "The projects were very relevant to the material learned in class and really challenged us to understand the concepts."

Successful Motivation

As discussed earlier, one of the primary motivations for this project domain was to engage the students and excite them. When the project was first discussed in class, reactions ranged from amusement to excitement, which was mostly sustained throughout the project. I heard a number of anecdotal stories about how the machine learning class was being called "The Mario Class" for the duration of the project. As students in the class continued working in the computer science lab, students outside the class begin commenting on the project and how it looked more like fun than work.

A more concrete measurement of motivation is that six of the nine students ended up doing more work on the project than was necessary. The project was designed so that there were multiple modules that could be completed (i.e., implement Monte Carlo and/or Q-Learning) which would earn them some number of points, based on quality and completeness. While part of their motivation was likely grade-related, these six students completed more modules than was necessary. We had not anticipated this when designing the project — while six students earning a grade of 100% on a project is impressive, future versions of the project will need to better distinguish between such high-performing students.

Project Extensions

A final measure of success and student engagement was that five of the nine students continued using the Mario domain for the course's third and final project. The third project was open ended, requiring students to propose a topic in machine learning that they could explore in a domain of their choosing. While some of the students were undoubtedly motivated to continue using the Mario domain in order to leverage their experience from the second project, it is doubtful they would have done so if they did not find the domain engaging.

The five students studied four different questions (two worked as a group), demonstrating the extensibility of the platform (related to Repenning et al.'s conditions #3 and #4). The variety in these topics helps demonstrate the flexibility of the Mario domain, and the potential for many exciting projects when using games in machine learning classes. The four projects investigated:

- **Meta Learning**: Can the agent *learn* to change its learning and exploration rates change over time, improving performance relative to static (hand-coded) rates?
- **Function Approximation**: Can a regression tree represent the action-value function, improving the agent's performance relative to a discretization of the action-value function?
- **Hierarchical Learning**: Can higher-level actions, such as "attack monster" and "retrieve power-up" be learned and improve performance relative to using only primitive actions?
- **User Shaping Reward**: Can a human user provide online reinforcement, in addition to that in the MDP, so that the agent learns faster than using only the reward of the MDP?

Conclusion

In this paper, we have argued that games should be increasingly incorporated into AI classes for pedagogical and engagement reasons. In the case study, we discussed how the barrier to entry was low for the game in question, and that other games may be equally appropriate for classroom use.

Furthermore, we provided anecdotal evidence that the use of such games could result in increased student engagement. We leave the important question of measuring the impact of games in upper level courses on student recruitment and retention to future work.

The next time this project is used in a course, we anticipate at least three changes. First, the hand-coded agent included with the distribution should be made simpler, so that students are more readily able to outperform it. Second, may be useful to conclude the unit with a final class-wide competition, allowing the students' different implementations to compete against each other. Third, the grading scheme should be changed to down-weight quantity: students who successfully complete more modules than required should not necessarily receive a high grade. We anticipate that the first two of these enhancements will further contribute to student motivation and satisfaction with the course, while the third will provide more accurate student assessment.

Acknowledgements

The author thanks Chun-wai Liew and the anonymous reviewers for useful comments and suggestions, John Asmuth for his prior work on the Mario domain, and Brian Tanner for his leadership on the RL-Competition.

References

Amato, C., and Shani, G. 2010. High-level reinforcement learning in strategy games. In *Proc. of the AAMAS Conference*.

Barnes, J., and Hutchens, J. 2002. Testing undefined behavior as a result of learning. In Rabin, S., ed., *AI Game Programming Wisdom*, Game Development Series. Charles River Media.

Bayliss, J. D. 2007. The effects of games in CS1–3. *Journal of Game Development* 2.

Bloom, B. S. 1956. *Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain*. Addison Wesley.

DeNero, J., and Klein, D. 2010. Teaching introductory artificial intelligence with pacman. In *Proc. of the EAAI Symposium*.

Kessler, R.; van Langeveld, M.; and Altizer, R. 2009. Entertainment arts and engineering (or how to fast track a new interdisciplinary program). In *Proc. of the SIGCSE Symposium*.

Mitchell, T. M. 1997. *Machine Learning*. McGraw-Hill.

Parberry, I.; Roden, T.; and Kazemzadeh, M. B. 2005. Experience with an industry-driven capstone course on game programming. In *Proc. of the SIGCSE Symposium*.

Repenning, A.; Webb, D.; and Ioannidou, A. 2010. Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proc. of the SIGCSE Symposium*.

Stanley, K. O.; Bryant, B. D.; Karpov, I.; and Miikkulainen, R. 2006. Real-time evolution of neural networks in the NERO video game. In *Proc. of the AAAI Conference*.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. MIT Press.

Whiteson, S.; Tanner, B.; and White, A. 2010. The reinforcement learning competitions. *AI Magazine* 31(2):81–94.

Wong, D.; Zink, R.; and Koenig, S. 2010. Teaching artificial intelligence and robotics via games. In *Proc. of the EAAI Symposium*.