Robot Defense: Using the Java Instructional Game Engine in the Artificial Intelligence Classroom

Scott A. Wallace
School of Eng. and Computer Science
Washington State University Vancouver
Vancouver, WA 98686
wallaces@vancouver.wsu.edu

Ingrid Russell
Department of Computer Science
University of Hartford
West Hartford, CT 06117
irussell@hartford.edu

Abstract

In this paper, we examine Robot Defense, a computer game that serves as a pedagogical platform for students to explore methods typically covered in an Introductory Artificial Intelligence course. Robot Defense is the synergistic outcome of two NSF funded Course, Curriculum, and Laboratory Improvement (CCLI) projects and was first presented in (Wallace, Russell, & Markov 2008). The primary contribution of this paper is to discuss the implementation of the Robot Defense platform and the outcome of its first use in the classroom.

Introduction

We believe, and research supports the notion, that students will be more engaged in their studies if their coursework focuses on interesting and relevant challenges (Renninger, Hidi, & Krapp 1992). This paper describes two Course, Curriculum, and Laboratory Improvement projects that share the goal of building compelling curriculum to teach traditional Computer Science course objectives.

The Java Instructional Game (JIG) Project (Wallace & Nierman 2006), began in 2006 as a collaboration between Washington State University Vancouver and the University of Puget Sound. The goals of the project are twofold. First, we aim to create a Java based game library that can be used throughout all four years of an undergraduate Computer Science program. Second, we aim to create a set of small projects that use JIG to teach traditional Computer Science learning objectives with games as the vehicle. The first version of JIG was developed in Summer of 2007 and used in the classroom later that year.

Project MLeXAI (Russell, Markov, & Coleman 2007; Kumar, Kumar, & Russell 2006) began in 2004 as a collaboration between University of Hartford, Central Connecticut State University and Gettysburg College. In this first phase, PIs at each of these institutions developed adaptable course curriculum and hands-on laboratory projects that could be closely integrated into traditional undergraduate Artificial Intelligence courses. Project MLeXAI's second phase began in fall of 2007 and involves 20 faculty members from around the country.

In the following sections of this paper, we begin by describing Robot Defense, the platform upon which our curricula is based. Next, we describe the planned first use of Robot Defense including a brief synopsis of three major projects. Not surprisingly, the actual use of Robot Defense differed from initial expectations in a few important ways; our discussion explores the actual assigned projects and the reasons for diverging from the planned syllabus. Finally, we summarize students' evaluation of the projects and compare the results to similar projects in MLeXAI. We conclude the paper with potential avenues for future work.

Robot Defense

Games are a conducive vehicle for Artificial Intelligence education; Youngblood, for example, documented that over 80% of the students in his AI for Interactive Computer Games Course were drawn in specifically due to the relationship with games (Youngblood 2007). With similar expectations that games may help increase student interest in Artificial Intelligence, Scott Wallace began implementing Robot Defense in late 2007 as JIG was being tested for the first time in the classroom. The Robot Defense platform consists of a simple real-time strategy game similar to the popular flash game Desktop Tower Defense. In the game, insects move across a tile-based map from a fixed source location to one or more destinations. Different types of terrain make this task more complicated as do the placement of special *fan* and *vacuum* towers that can influence the path of the insects—possibly preventing them from reaching their destination. Fans and vacuums, of course, require power; and power in this game is a resource that must be procured by collecting crystals that appear randomly on the map. Robots travel over the terrain subject to the same constraints as the insects and these robots are tasked with gathering the crystals which power the fans and vacuums. Thus, the game entities can be viewed as two opposing teams: on one hand the insects trying to traverse the map; on the other hand the robots and towers trying to prevent this from happening. While the game bears some similarities to the classic Wumpus World (Russell & Norvig 2003) its multi-agent, dynamic, and fully observable characteristics are all critical differences from the standpoint of agent design.

Robot Defense is implemented in Java. In Java, objects are discovered and loaded at runtime, and this makes

it relatively straightforward to create applications in which some of the requisite objects only become available after the application has been distributed. Robot Defense (as well as a number of other JIG projects) leverages this property through the Java Service Provider Interface (Seacord & Wrage 2002). Under this model, the application (here the Robot Defense game) defines one or more interfaces or abstract classes which serve to declare the signature of a particular *service*. A path planning service, for example, may simply be described by a single method which takes three arguments: a graph; a starting node; and an ending node. The method may then be declared to return a list of edges, which when traversed in order will move a visitor from the start node to the end node. With such an interface, one could then define specific search *service providers* (implementations) which perform depth first search, A* search, etc.

The beauty of the Service Provider approach is that the application can be designed, implemented, and then distributed as a jar archive using only the interfaces to describe the services. The assumption is that when run, one or more service providers will be identified that allow the application to perform its necessary function. Robot Defense is compiled with a variety of service definitions. Each service corresponds directly to an assignment (such as path planning) that the student will later implement. When run, Robot Defense looks for the student's implementation and dynamically loads it into the game. The student need not rebuild the game or the jar. However, the student's service providers must be recognized by the Java runtime. The standard method of registration relies on a set of text files stored in a `META-INF` directory located on the class path. Simply by modifying the contents of these text files, student can incorporated their own program segments into the Robot Defense game.

Figure 1 illustrates a typical directory structure for a Robot Defense distribution. The jar archive sits at the same level as the student's code (here named `StudentAStar.class`) and at the same level as a `META-INF` directory that contains an entry for the `jig.rd.Pathing` service. This text file itself contains the full name of the service provider. The game would then be launched by adding the current directory and the jar file to the class path and calling the entry point associated with the assignment (here `jig.rd.PrjPathing`)[1].

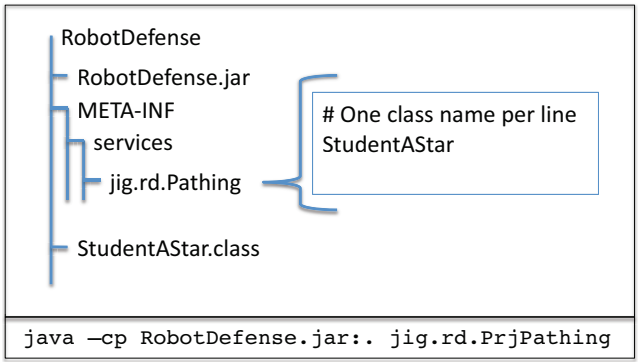

Figure 1: Directory structure (top) and Java command line (bottom) for a typical JIG curricular distribution.

Planned Syllabus

The anticipated project syllabus (described in (Wallace, Russell, & Markov 2008)) contained four projects built on the Robot Defense platform. In the first two projects, students would implement two search algorithms to provide the path planning that would allow autonomous robots to gather randomly appearing resources from the map and to exit the map from any location. The first task would require the use of A* search, whereas the second would be accomplished via Best First Search/Dijkstra's algorithm.

In the third project, students would focus on the logic used to control the vacuum and fan towers. Using the Soar agent architecture (Laird, Newell, & Rosenbloom 1987), students would create a set of rules that govern the behavior (rotation and power level) of towers to efficiently capture insects and thereby prevent them from reaching their goal. The task is complicated because the way in which insects are affected by vacuum/fan air currents is based on an internal aerodynamic function that the agent does not know. This function takes into account all of the air currents acting on a particular insect and translates that into a force that is applied to redirect the insect's motion. Thus, different aerodynamic functions can, for example, make it more likely that the insect will move: if there is more than one tower acting on the insect; if the air currents are pulling all in the same general direction; if the air currents are pulling in a specific direction (e.g., west); or if the insect is of a particular type (e.g, beetle vs worm). Encoding a relatively good strategy for one or two of the aerodynamic functions is relatively straightforward, but it is not a simple matter to encode a fixed strategy that will work well with all possible aerodynamic functions.

The fourth assignment aimed to address the difficulties associated with creating a strategy that would perform well under a variety of aerodynamic functions. Instead of creating a fixed strategy using Soar, students would implement an agent that obtained observations directly from the Robot Defense environment and used these observations to learn a correct response to whatever aerodynamic function was in place at the time.

Actual Syllabus

Shortly after the Spring 2008 offering of the Introduction to Artificial Intelligence course was under way, it became clear that a few modifications to the original assignments would be needed as students were not progressing through the materials as quickly as the instructor had anticipated. The actual course syllabus therefore diverged somewhat from the initial plan. In this section, we discuss how the Robot Defense platform was actually used in the course and provide the reasons behind our modifications.

[1]Note that the service and project names specified here are for illustration; they have been shortened for clarity of presentation.

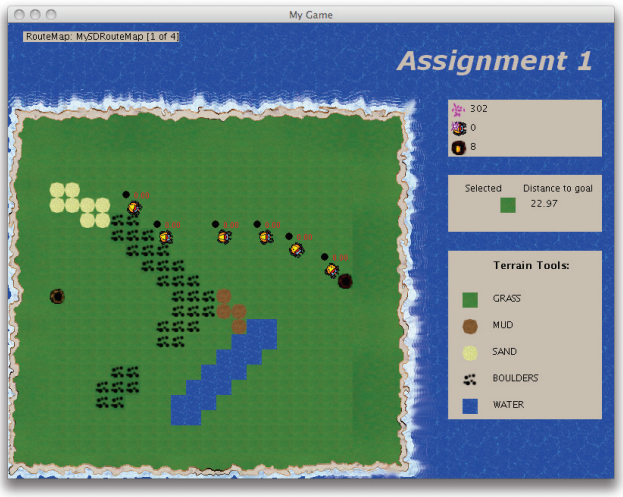

Figure 2: Insects travel on the shortest cost route from their source to their destination in the Robot Defense Dijkstra's Algorithm project.

Project 1: Dijkstra's Algorithm

Figure 2 shows a screen capture from the first project in which students implement Dijkstra's algorithm. In Robot Defense, the map is internally represented as a graph in which each 20 by 20 pixel tile corresponds to a node and adjacent nodes (in cardinal and diagonal directions) are connected with edges. The edges themselves are weighted based on the terrain types of the tiles to which they are connected. Each terrain has its own cost—the least expensive being grass (cost 1) and the most expensive being water (infinite cost); sand, mud and boulders represent intermediate costs in increasing order.

As with all of the projects built on the Robot Defense platform, students implement a specified service provider that is incorporated into the game at run time. The game provides user interface controls to switch service providers during play. This allows students to compare two implementations with each other, or to switch between a default (stub) implementation that is provided by the instructor and their own fully featured implementation.

Implementation In the Dijkstra's project, students are provided with the Robot Defense jar archive and implement a provider of the `SDRouteMap` (Single Destination Route Map) service. The service contains three important methods: `cache(Graph, List)`, `costFrom(Node)` and `directionAt(Node)`. The `cache()` method performs a one-time calculation (here Dijkstra's algorithm) to compute the best route from any location on the graph to any one of the destination nodes. The results of the calculation are then stored internally in the student's service provider and accessed on demand by the Robot Defense game using the two other methods (`costFrom()` and `directionAt()` which indicate respectively the cost of the optimal path from the current node to the destination and what edge to follow as the next step on that path.

This first project comes with a default provider of the `SDRouteMap` service that is built into the Robot Defense jar. This stub provides no useful information; rather the `directionAt()` method returns a random outgoing edge and the `costFrom()` method returns -1 for all nodes in the graph. Thus, at the onset of the assignment the insects have no directed way of moving from their start location to their destination. Instead they simply take a random walk without regard to the cost of different tiles or the relative direction of their goal. Students can switch between this service provider and their own by clicking the *Route Map* button in the upper left corner of the UI. In this manner, Robot Defense aims to provide a source of visual feedback that both indicates why the algorithm is useful and important and also whether the student's implementation is reasonable and correct.

Discussion Overall, the first assignment was well received by the students. However, it was also clear that because the course involved both senior-level and junior-level students there was a wide range in programming proficiency, especially with regard to the Java language (in some cases, the juniors may have had as little as one semester of previous experience with the language). Based on the instructor's observations, there seemed to be two main sources of difficulty for the students.

The first difficulty for the students was to understand the context in which their code was being used. Specifically, for many junior-level students working with an external environment such as Robot Defense is a new challenge. Unlike tasks in which students write all the code themselves, here students are forced to understand something about the interactions between their own code and the environment in order to make progress on their task. Of course, as Computer Scientists, this is a critical skill to acquire, but it is also one that students may not immediately appreciate as it often seems irrelevant to the task at hand.

The second common difficulty students had was understanding the use of Java generics. While most of the seniors had experience with generics, the concept and certainly the syntax was new to many of the juniors. Although this was probably a relatively minor hurdle, it did provide one more obstacle for students to overcome before they could get to the task of implementing Dijkstra's algorithm.

In future iterations of this assignment, there are at least two possible ways in which these hurdles may be dealt with. Instructors wishing to focus exclusively on the AI concepts, may wish to provide very complete documentation for how the student's code will be used by the system and how all interactions will take place between the library code and the student's own code. This would likely alleviate many of the issues surrounding the first difficulty. Generics could also easily be removed from the Robot Defense service interface either by modifying the game itself or by providing a wrapper for service providers. However, while some instructors may wish to reduce these hurdles, others may opt instead for an experience in which students have to deal with some of the challenges associated with API programming and programming in the large. It is our opinion that students will be open to these challenges *if* the instructor identifies them

Figure 3: A robot plans an optimal path (indicated with red squares) to crystal resources in the Robot Defense A* Search project.

as important learning objectives at the beginning of the assignment. Otherwise, it seems likely that many students will only view these as obstacles to the assignment's true learning objectives.

Project 2: A* Search

The second project followed directly after the first. Here, students implemented A* search to guide an autonomous robot from its current location to resources that periodically appear at random locations on the map. Figure 3 presents a screen capture of the robot following its planned path. The assignment was separated from Project 1 mainly because the use of A* and Dijkstra's algorithm within the Robot Defense game are quite different. Whereas Dijkstra's algorithm is used to compute and cache path information for any location on the map to the goal destination; A* is used dynamically to plan point to point paths where the start and end location change between each search. Because it does not make sense to cache information after A* has completed, the interfaces that describe the services for these two projects are relatively different.

Implementation In the A* project, students are provided with the Robot Defense jar archive and implement a provider of the `SearchFactory` service. The service contains two methods which follow the pseudo code outlined in (Russell & Norvig 2003): `initializeProblem(Node, Node, Graph)`; and `aStar(SearchProblem)`. The first method returns a `SearchProblem` instance which is used to encapsulate the information required by any graph-based search routine. The starting node, destination node and the graph to be operated on are all parameters to this method. Once packaged, the search problem instance is then sent to a particular graph search implementation (here `aStar()` which in turn returns a `SearchNode` object that can be used to trace lowest cost path from start node to goal node.

Discussion As with the Dijkstra's implementation, a number of students struggled with basic Java concepts in this project. The struggles were somewhat heightened because the pseudo code of Russell and Norvig requires students to implement three objects: a `SearchNode`; a `SearchProblem`; and a `SearchFactory`[2]. While this design can yield highly reusable code, it was not clear how much students valued the generality of the design when weighed against the intellectual overhead associated with determining how all the objects were supposed to integrate with one another.

Many instructors (including ourselves) believe that visualization can be an important learning aid. However, it is interesting to note that even though students were given a visual environment that provided direct feedback by displaying the path returned by the A* algorithm, not all students submitted successful solutions. Indeed, $6/17$ or 35% submitted A* implementations which were clearly broken in that careful consideration of the visual feedback would have alerted the astute observer that the algorithm was not returning optimal paths. This suggests that some students are either: 1) failing to critically analyze the feedback they are given, even when it is visual in nature; or 2) ignoring their analysis as a result of other factors (lack of time, etc). Thus, an important follow up would be to ask students when they submit their work to judge the quality of their implementation. One might then be able to compare responses on visual and non-visual versions of the same assignment and determine what, if any, impact the visualization had on the students' submissions.

Project 3: Soar Agents

In the original syllabus, students' next project would have been to implement control knowledge for the vacuum and fan towers using the Soar agent architecture. This course project was incorporated into Robot Defense as planned, but was later removed from the syllabus. Below, we briefly discuss the motivation behind this choice.

Discussion The 2008 offering of the Introduction to AI course was the first time that Soar was scheduled for use in the WSUV undergraduate classroom. The original syllabus allocated six lecture hours (two weeks) for a discussion of Soar; this discussion was planned to finish with the Soar agent project.

While the ideas behind Soar and other rule based systems are conceptually simple, the execution model presents a significant paradigm shift for students. This means that programming even simple behavior can be quite labor intensive. As the course developed and the lectures on Soar began, it quickly became clear that six hours of class time and one assignment would not be sufficient for students to make the intellectual leap to understanding rule based systems. After considering the students' progress, the instructor decided

[2]Note that while the `SearchFactory` is not explicitly indicated in the Russell Norvig pseudo code such an object must exist in Java to provide the container for the actual search method.

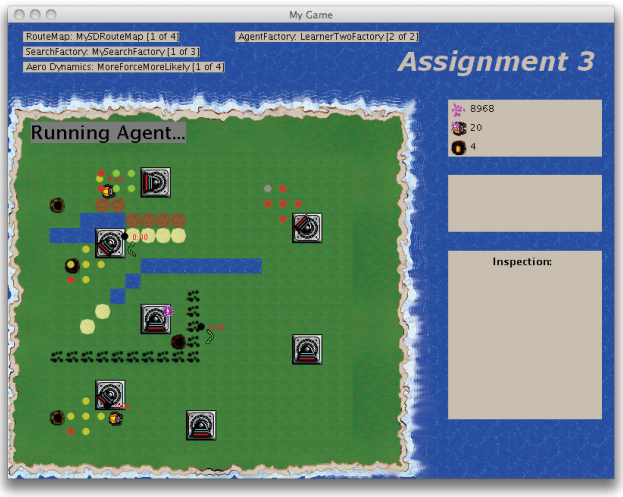

Figure 4: Q-Learning modifies the Vacuum Tower's response function.

to make two changes: first, course time allocated to Soar and Soar programming was extended by another week and a half; second, students were assigned a Soar project based on the Eaters environment instead of Robot Defense. The motivation for this second change was simple: like Robot Defense, Eaters is a game-like environment. However, the critical difference is that Eaters comes pre-packaged as a part of the Soar tutorial so there were substantial learning materials and references built into the Soar documentation to support the student's learning process. Moreover, because Eaters had been in active use for many years, there were less concerns that students would be waylaid by bugs in the environment.

In the post-hoc analysis, it seems clear that for students to have a successful experience with Soar, at least four weeks, and perhaps as many as six weeks would need to be allocated to the architecture. While we believe that students profit greatly from familiarity with a range of different programming paradigms, a six week commitment may not be feasible for many instructors of the Introductory Artificial Intelligence Course.

An alternative approach that we may pursue in our next offering is to let students choose how to create their own hand-coded agent. Ambitious and interested students may use the Soar architecture while less ambitious students would be able to use the Robot Defense agent interface classes to implement the agent's logic procedurally using the Java language itself. Both methods serve to expose knowledge representation issues and to lay the groundwork that would later highlight the value of learning.

Project 3′: Q-Learning

The final Robot Defense project was given in the last third of the semester as the course examined machine learning algorithms. In this project, students implemented Q-Learning so that Robot Defense vacuum towers could learn a response function to capture insects regardless of what aerodynamic function is being used by the game at runtime. Figure 4 presents a screen capture of the game with the towers being controlled by a Q-Learning agent. Over time, the agent becomes more adept at capturing insects and also learns that it is valuable to turn off power when there are no insects near a particular tower.

Implementation In the Q-Learning project, students are provided with the Robot Defense jar archive and the following critical skeletal code: a `StateVector` implementation that generates a usable and useful representation of the state near any specified tower; a `LearnerOne` agent implementation that provides a working (but relatively uninteresting) agent that associates states with actions. The student's task is three fold: 1) to assign meaningful rewards for events that happen in the world (such as capturing insects, or using resources); 2) implement Q-Learning to map states to actions; 3) ensure that the agent explores all actions that may be relevant to the situations it encounters. The environment produces a log file that indicates how many insects have been captured and how many resources have been used over the course of the game. By comparing these files (or a graph of their results) students can see how differences in their learning algorithm or internal representation affects the agent's overall performance.

Discussion The Q-Learning project required somewhat more code than either of the two earlier Robot Defense projects. However, the challenges faced by students were ameliorated by two facts. First, by this point in the semester, third year students had become increasingly adept at Java programming and were on more equal footing with the fourth year students. Secondly, unlike the previous projects, the instructor provided skeletal code with this assignment that gave students a foothold on how to begin. Given their improved experience and the skeletal code, nearly all the students in the class were able to create agents that successfully learned how to capture insects under a variety of aerodynamic functions...a task that would have been extremely difficult using a solely hand coded approach. Moreover, because most students' implementations were able to learn an appropriate response function over the course of approximately a minute, students could easily see their agent's performance improving while they watched the game unfold.

Evaluation and Students' Perceptions

At the end of the course, all students in the course were asked to complete a survey regarding the Robot Defense projects. Sixteen of the 17 students attended class the day of the survey and submitted a response. Below, we highlight some of the most salient questions regarding the student's perceptions of the learning experience.

As noted in the previous sections, from the instructor's perspectives, students were most challenged by two aspects of the Robot Defense projects. The first was use of relatively sophisticated Java syntax such as generics. The second was a challenge common to any large programming task: that students were not in complete control of the entire code base, and thus had to program to an interface that

Question	(Agree or Strongly Agree)	
	Robot Defense	MLeXAI Phase 1
The time allowed for project was sufficient	81%	75% - 100% [*90%*]
The project was interesting to work on	94%	75% - 100% [*84%*]
The project contributed to my overall understanding of the material	100%	83% - 100% [*95%*]
I had a positive learning experience in this course	88%	75% - 100% [*94%*]
The student project took a reasonable amount of time to complete	81%	50% - 100% [*82%*]
Based on my experiences in this course, I would like to learn more about AI	94%	71% - 100% [*85%*]

Figure 5: Student Perception on Robot Defense

was pre-specified and may not have been designed in a way that was most intuitive for that particular student. These challenges are echoed in the written comments where students were asked what they liked least about the projects (ten responses): 3 students found the projects generally confusing; and 3 others specifically found the documentation inadequate.

Overall, however, many students were quite positive about the project. When asked what they liked best (nine responses), six indicated that they found the project fun, enjoyable, or interesting and three of these specifically indicated that they valued being able to visualize the results of their work. Students were also quite positive in their responses to Likert scale questions about the projects. Salient responses are illustrated in Figure 5. Column 2 indicates the percentage of the sixteen respondents marking *agree* or *strongly agree* while the third column indicates the same percentage as a range over all Phase 1 MLeXAI projects (the average value over all projects is italicized in square brackets). Average student responses fall solidly within the range reported by other MLeXAI projects. Students do, however, seem to have had a positive experience with the Robot Defense projects despite their challenges. In particular, more students agree that the project was interesting vs. the average Phase 1 MLeXAI project (row 2, 94% vs 84%), more students agree that the project contributed to their understanding (row 3, 100% vs 95%) and more students indicated they would like to learn more about AI (row 6, 94% vs 85%). Interestingly, when comparing Robot Defense to other MLeXAI Phase 1 projects and classroom experiences, more students in the WSUV Spring 2008 course agreed that their project (Robot Defense) was interesting (94%) but fewer seemed to be in agreement that they had a positive learning experience (88%). This suggests that some of the tasks and assignments outside of Robot Defense may have been responsible for the lower level of agreement about a positive learning experience.

Conclusions and Future Work

We have described the current version of Robot Defense, a game based platform for presenting a variety of topics typically covered in an Introductory to Artificial Intelligence course. The initial use of this platform was generally well received by students and comparable to many of the other projects built for AI coursework under Phase 1 of the MLeXAI project. Phase 2 of the MLeXAI project, which Robot Defense is a part of, will further investigate the impact of theme and problem based learning on student outcomes and examine whether the results of Phase 1 can be extended and generalized by 20 faculty across the country.

The Robot Defense project described here does highlight a few remaining interesting questions, most notably the question of whether visualization benefits learning outcomes. Currently, the JIG team is developing a variety of curricular projects to be used in smaller colleges and universities across the Pacific Northwest as part of the Northwest Distributed Computer Science Department. We expect that this community will be able to bring together enough students and faculty to examine this question along with a number of others.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation under Grant Numbers DUE-0409497, DUE-0716338, DUE-0633726 and CNS-0829651. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References

Kumar, A.; Kumar, D.; and Russell, I. 2006. Non-traditional projects in the undergraduate ai course. In *Proc. of the Annual SIGCSE Technical Symposium on Computer Science Education*.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *AI* 33(1):1–64.

Renninger, K. A.; Hidi, S.; and Krapp, A. 1992. *The Role of Interest in Learning and Development*. New York, NY: Lawrence Erlbaum Associates.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition.

Russell, I.; Markov, Z.; and Coleman, S. 2007. Project mlexai: Applying machine learning to web document classification. *J. of Computing Sciences in Colleges* 23(2).

Seacord, R., and Wrage, L. 2002. Replaceable components and the service provider interface. Technical Report CMU/SEI-2002-TN-009, Carnegie Mellon (http://www.sei.cmu.edu).

Wallace, S. A., and Nierman, A. 2006. Addressing the need for a java based game curriculum. *J. of Computing Sciences in Colleges* 22(2):20–26.

Wallace, S. A.; Russell, I.; and Markov, Z. 2008. Integrating games and machine learning in the undergraduate computer science classroom. In *Proc. of the 2008 Conference on Game Development in Computer Science Eduation*, 56–60.

Youngblood, M. 2007. Using XNA-GSE game segments to engage students in advanced computer science education. In *Proc. of the 2007 Conference on Game Development in Computer Science Eduation*, 70–74.