# Teaching Introductory Artificial Intelligence through Java-based Games

**Amy McGovern**
School of Computer Science
University of Oklahoma
amcgovern@ou.edu

**Zachery Tidwell**
School of Computer Science
University of Oklahoma
ztidwell@ou.edu

**Derek Rushing**
School of Computer Science
University of Oklahoma
Derek.M.Rushing-1@ou.edu

**Abstract**

We introduce a Java graphical gaming framework that enables students in an introductory artificial intelligence (AI) course to immediately apply and visualize the topics from class. We have used this framework in teaching a mixed undergraduate/graduate AI course for six years. We believe that the use of games motivates students. The graphical nature of each game enables students to quickly see how well their algorithm works. Because the topics in an introductory AI course vary widely, students apply their algorithms to multiple game environments. A final challenging environment enables them to tie together the concepts for the entire semester.

## Introduction

Our goals in teaching introductory artificial intelligence (AI) are to create a significant learning experience (Fink 2003) and to enable students to develop the problem solving and teamwork skills necessary for success in their future career (Woods et al. 2000). These goals are intertwined. To achieve them, students need to be deeply engaged in the subject matter. Inspired by past success in teaching AI (McGovern and Fager 2007; DeNero and Klein 2010), we have created and tested a general gaming framework for projects for an AI class. We have been teaching a mixed upper level undergraduate and graduate introductory AI course for six years. Because such a class covers a wide range of topics, using a single project paradigm for the entire semester can be difficult. We have found that the use of graphical game-based projects keeps the students' interest level high and engages them in the projects. In this paper, we introduce a general Java[1] gaming framework that enables us to use a variety of games and enables students to quickly visualize their agents. We assume students are either familiar with Java (through required undergraduate courses) or capable of quickly learning Java (expected of graduate students).

Fink defines a taxonomy of significant learning that builds on the traditional Bloom's taxonomy of learning (Bloom 1956). In Fink's work, a course that becomes a significant learning experience should integrate six dimensions of learning: 1) foundational knowledge, 2) application, 3) integration, 4) the human dimension, 5) caring, and 6) learning how to learn (Fink 2003). Traditional course lectures and exams focus on foundational knowledge. In our teaching, we use active learning to engage students (Prince 2004; Felder and Brent 2009). We use the projects to focus on all six dimensions of significant learning.

Specifically the projects 1) enable students to better remember their foundational knowledge by 2) applying it to new domains. By applying the techniques to a variety of domains, they 3) integrate their knowledge and connect ideas from previous courses. By working within an existing codebase, they demonstrate and modify their own skills while working with other students which addresses 4) the human dimension. Although we do not specifically address caring 5), students gain a new appreciation for AI by participating in the class and projects and learn that they are interested in it for some aspect of their career. By working on projects across a wide variety of domains and skills, we enable students to learn about material more deeply and become better students. And, we encourage creativity through extra-credit, which motivates many of the students to learn additional material about each topic 6).

In addition to the dimensions of learning specified above, Fink also specifies that a successful learning experience for students includes high quality and frequent feedback. The graphical nature of the projects provides this because students are able to immediately observe the higher-level behavior of their agents. It also enables them to quickly debug their agents as they refine their overall understanding of the specific topic.

Computer games have a long history of motivating learning both outside computer science (for a thorough treatment of this topic, see Gee, 2007) and within computer science (e.g. Alice[2], Scratch[3], introductory programming courses such as Leutenegger and Edgington, 2007 and Bayliss, 2009 describe, and cross curriculum programs such as Burns, 2008 and Sung 2009). We focus specifically on using games in teaching AI, a topic discussed at both the 2010 and 2011 Symposium on Educational Advances in Artificial Intelligence (EAAI).

[1]http://www.java.com

[2]http://www.alice.org/

[3]http://scratch.mit.edu/

## Course goals and details

The learning objectives for our introductory AI course are as follows:

- Select the AI technique best-suited for a novel problem/domain and justify your choice, including an analysis of the complexity of your choice
- Implement an AI solution to a complicated real-world problem and evaluate its effectiveness
- Gain the skills, confidence, and experience to implement your own AI solutions within an existing large codebase
- Function effectively in a team

The first and second objectives directly relate to a project-oriented class. By giving the students experience at implementing the AI techniques in a variety of environments, they can better understand why they should choose a particular technique for a problem domain and they can immediately see how effective it is. Although the third and fourth objectives do not directly relate to the topic of AI, they are critical for our goal of enabling students to develop problem solving and teamwork skills that will be useful throughout their career. We provide an existing codebase for each of the games and students must implement their agents within this codebase. For most of the students, this is their first experience in working with a large existing codebase rather than implementing their project from scratch, as is traditional in many classes. By the end of the semester, the students have gained critical confidence in their skills as computer scientists. For the final objective, we have experimented with a variety of team-based projects over the six years of teaching this course. We believe that teamwork is an essential skill for a computer scientist and we strive to help students gain skills in this arena by working together.

We divide the topics covered in our course into three thematic areas: search, learning, planning/multi-agent systems (planning overlaps multi-agent systems). We use Russell and Norvig's book (Russell and Norvig 2009) as the course textbook.

We next describe each of the games, the projects corresponding to each environment, and the AI topics covered by each of the projects.

## Asteroids/Spacewar

Inspired by our past success of using Spacewar to teach AI (McGovern and Fager 2007), we began our games with the Spacewar source code[4]. Spacewar is a reimplementation of the classic arcade game of asteroids (Graetz 1981). In the original game, a single ship attempts to navigate safely around an asteroid filled environment and to shoot the other ships. We have modified this to remove the effects of gravity and the single central sun. Instead, the environment contains some number of (potentially moving) obstacles and other ships. Each ship carries an energy cell that fuels its thrusters, shields, mines, cannons, electromagnetic pulses, tractor/repulser beams, and life support systems. If the energy cell is depleted, the ship self-destructs. A ship can

[4]Available open source at http://code.google.com/p/spacewar/

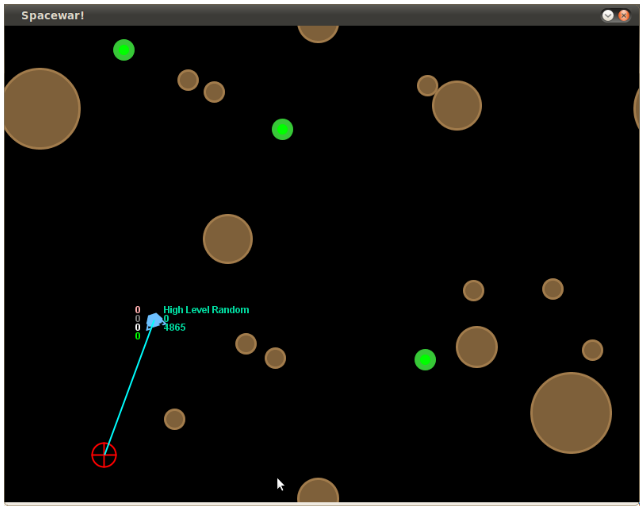


*Figure 1: A single agent moving around the Spacewar environment. The brown circles are asteroids of varying diamters. The green circles are energy beacons. The agent is aiming for the red circle.*

recharge its energy cell by collecting energy beacons from the environment, or by returning to its home base if it is available. Ships can also be damaged or destroyed by collisions with asteroids floating in space, from the fire of other ships, or from mines. They can temporarily invoke a shield to protect themselves but the shielding costs energy. Ships become temporarily uncontrollable if they are hit by an electromagnetic pulse. Asteroids can be damaged or destroyed by shooting at them. Our original implementation used only the features noted in (McGovern and Fager 2007). However, students continued to request additional features and developed many of them on their own.

Spacewar is an interesting arena for both AI and machine learning algorithms. Using terms from (Russell and Norvig 2009), it is partially observable, stochastic, sequential, dynamic, continuous, multi-agent, and competitive. This makes it a challenging problem for any AI project. Successful agents will avoid collisions as well as shoot opponents. These individual tasks are difficult and combining them makes the task even more challenging.

The Spacewar system is quite flexible and enables students to work on a variety of projects. We have successfully used it for introducing A$^*$ navigation in a more realistic environment than is typically presented in textbooks, for learning using both genetic algorithms and reinforcement learning, for classical planning, and for multi-agent coordination.

Because A$^*$ search is designed for fully observable, deterministic, episodic, static, discrete, single-agent environments (exactly the opposite of Spacewar), students must stretch their understanding of A$^*$ to learn how to apply it to a more real-world task. Figure 1 shows a single agent Spacewar environment where students program their A$^*$ agent to navigate safely around asteroids without the interference of another agent. This provides a challenging initial project. For the A$^*$ navigation project, we give the students the following instructions and guidance. As with our code, the complete assignment (including rubric) is available at the URL specified at the end of the paper.

*This project will use $A^*$ search for navigation. However, because the environment is continuous and the obstacles and ships are moving about the environment, you will not be able to use straightforward $A^*$ search. To deal with the continuous environment, you should implement ONE of the following changes to $A^*$.*

- ***Gridded $A^*$:*** *Break the environment up into n grid squares (where you determine the grid spacing). Create a graph based on these grid squares where adjacent grid squares are connected unless one of them contains an obstacle or a ship. Implement $A^*$ search to traverse the graph and find a path from the starting state to the goal beacon. You must design your own admissible heuristic.*
- ***Hierarchical gridded $A^*$:*** *The idea behind this approach is similar to the gridded $A^*$ except that the grid spacing is not fixed. Instead, the agent searches at a very coarse granularity (large grid squares) and finds a path in this easier space. The agent then refines its path by making finer grid squares along the previous path. This process is refined until the agent has a reasonable path from the start state to the goal state. As with the previous approach, this approach will require you to design your own admissible heuristic.*
- ***Roadmap $A^*$:*** *This approach is based on roadmap navigation. Instead of making grid squares of your entire environment, sample $n$ points from the space and draw a straight line between each set of points. If the line connecting the two points does NOT intersect an obstacle or ship, connect the two points in your search graph. Implement $A^*$ and search on this graph. As with the previous approaches, this approach will require you to design your own admissible heuristic.*

*Any one of these approaches will allow you to create a navigation plan in a continuous environment using $A^*$. However, none of these approaches will allow you to deal with the dynamics of the environment. This is inherently a tricky problem and the most straightforward way to deal with the dynamics is to replan frequently.*

The Spacewar environment is flexible enough to design a variety of games and challenges and we have also used the capture-the-flag scenario described by (McGovern and Fager 2007). In this scenario, the ships compete in teams of three to capture the enemy team's flag and bring it back to base. The winning team is the one that captures the most flags within five minutes. The environment we used for these games is shown in Figure 2. Here, the asteroids are not moveable but the other aspects of the game remain the same and the agents must be careful not to shoot members of their own team. This year we added a key to the game. Each team must capture the key and unlock its base before it can deposit flags.

The following is part of the instructions for the capture-the-flag assignment. This assignment explicitly integrates multi-agent system coordination and classical planning.

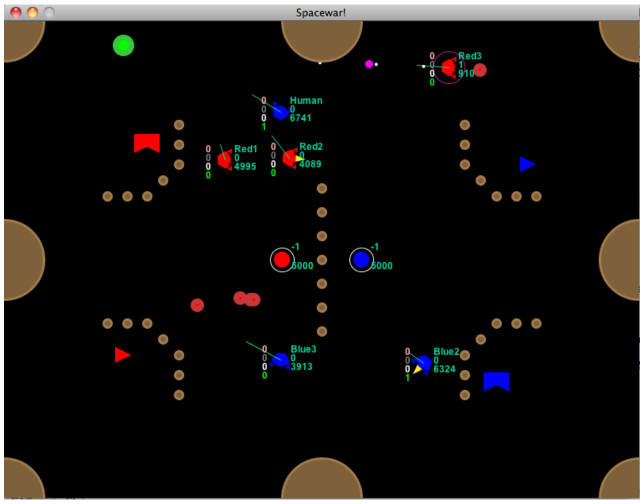


*Figure 2: An example of a team task in the Spacewar environment. Here the red team and the blue team are playing capture the flag. The small purple circle (upper right) is an electromagnetic pulse shot by one of the red agents. The red circles represent mines. The white dots are cannon fire. The sawtooth represents the key and the triangle represents the flag.*

It implicitly also integrates the topics they have learned throughout the course. This year, we initially implemented $A^*$ in Pacman, as described later in the paper.

*Your task is outlined below. You will be using planning for part of the project as well as integrating many of the other techniques you have learned this semester. We will do some of the group work in class.*

- *First, create an agent that can move around the environment intelligently. You can use the provided $A^*$ code (see implementation details for more information on this) for this or you may write your own. This agent should avoid the asteroids and move efficiently from one location to another. This will form the basis of many of your high level behaviors.*
- *Identify a set of high-level behaviors appropriate for each capture-the-flag team team member. Implement the set of high-level behaviors designed by your group. You are not constrained by a specific method for implementing your agent but you should use what you have learned so far this semester. Heuristics, search, and learning are all appropriate tools in your toolbox! Try to make effective use of the many types of weapons available to your ships as well as good use of the avoidance strategies.*
- *Implement planning within your team or within a single agent. I suggest that you implement it at the highest level (e.g. deciding which high-level behavior to choose at any time) but you could also use planning at a lower level such as improving $A^*$ or other navigation functions or in deciding which weapon to use in a given situation. For each behavior used by your planner, specify a set of pre-conditions and post-conditions, appropriate for use in*

*planning.*

- *Implement a central command agent that does effective multi-agent coordination. If you used planning at the highest level, this can be done using planning. Ensure that you re-plan as frequently as events warrant, such as a high-level action completing or sudden loss in energy. If you put planning at a lower level, then ensure that your agents are effectively coordinating their actions using another AI approach.*

For several years, we used only Spacewar-based projects in our class but we found that some of the projects were not as well suited to the topics we wanted to cover. Although $A^*$ navigation in a continuous environment gives students a better idea of how to apply the techniques to realistic problems, it proved difficult for many students for an initial project. Similarly, the genetic algorithms match the domain well but took so long to actually learn a policy that students would run out of time to investigate different parameter settings for their learning agents. One of the difficulties with Spacewar that we have been trying to mediate is that making an agent that can successfully navigate (and shoot other agents with accuracy) requires a substantial amount of background material beyond the scope of the class. For example, although we provide the students with proportional-derivative controllers (Craig 2004) to enable them to control the acceleration of their ships, many of them struggle with the basics of having the ship follow a path. Although we strongly believe the Spacewar domain is a valuable one for AI, the students' initial struggles with it guided us towards identifying simpler domains for the initial projects. This year, we reserved Spacewar for the later projects, which integrate much of the knowledge from the class into an overall intelligent agent. Since the environment is more realistic, it provides an opportunity to integrate their knowledge from the semester into a challenging final project.

## Pacman

Inspired by (DeNero and Klein 2010), one of the authors (who was a student in our machine learning class at the time) wrote his own Java Pacman simulator. We integrated it into the same Java gaming framework that we used for Spacewar and we used it for the the initial AI projects this semester.

Pacman is a simpler environment than Spacewar. Using terms from (Russell and Norvig 2009), it is fully observable, deterministic (although the ghosts provide a stochastic nature to the outcome), episodic, static, discrete, and either single-agent (if you ignore the ghosts) or multi-agent (if you pay attention to the ghosts). This makes it easier to apply traditional search methods. Our theory was that a discrete environment with simpler dynamics would provide a conceptually easier framework for student's initial search implementations.

Our implementation of Pacman follows the same rules as the traditional arcade game[5]. Pacman's job is to eat as many pellets as possible while staying away from ghosts.

[5]http://en.wikipedia.org/wiki/Pac-Man

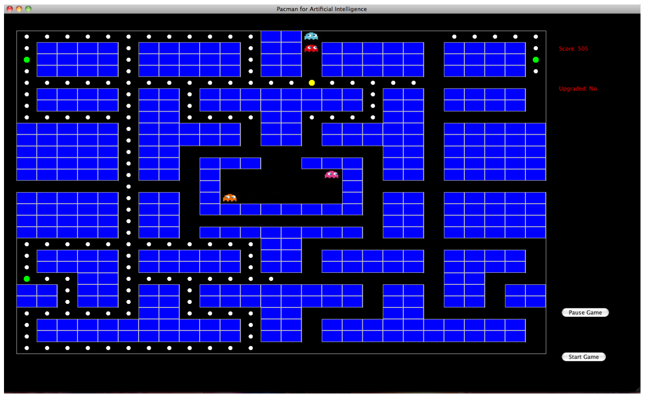


*Figure 3: An example of a pacman board with pacman part-way through its search on the board.*

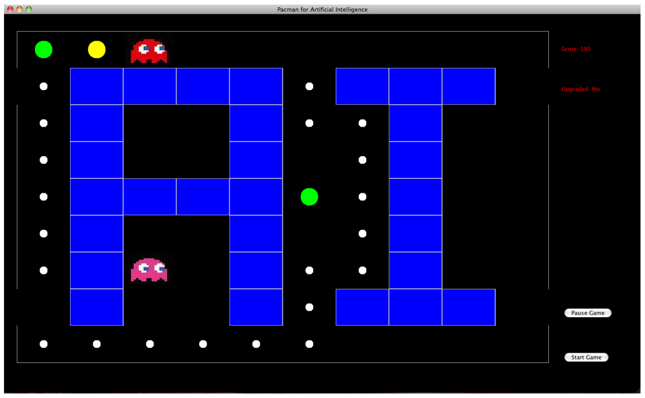


*Figure 4: An example of a student designed pacman board.*

The board shown in Figure 3 is modeled on the original Pacman arcade board but students are able to design their own boards, such as the one shown in Figure 4. The boards contain power pellets, which enable Pacman to temporarily eat the ghosts. In addition, special high point items appear on the board periodically, similar to the fruits that would appear in the arcade version of Pacman.

We used Pacman for uninformed and informed searches. For their first project, students implemented depth-first-search and breadth-first-search. For the second project, students implemented $A^*$ and hill climbing searches. For each search technique, students were given instructions to intelligently choose a location to send Pacman to and then to use the search technique to determine how Pacman should arrive at that point. They were encouraged to use other techniques or heuristics to decide where to send Pacman. For example, one strategy could be to aim for power pellets when Pacman is not in the powered mode and then to eat as much of the board as possible while in the powered mode. A student could achieve an "A" project by implementing the search technique correctly but the student could also potentially receive extra credit by creating a Pacman agent that received a consistently high score by eating pellets and ghosts. The extra credit was awarded from the class-wide competitive ladder, discussed below.

Although any of the games we present in this paper can be used for learning, this year we used Pacman for the genetic algorithms. We have successfully used both Spacewar and Roborally (discussed below) for both genetic algorithms and reinforcement learning. However, due to the complicated nature of those games, learning typically takes a considerable amount of time, which can frustrate the students. This year we had the students use evolutionary computation to make an intelligent Pacman agent. This project required the students to design an abstract state space and a set of high-level actions. The following is an excerpt from their assignment.

*For this project, your task is to use genetic algorithms/evolutionary computation to create an intelligent Pacman agent. Your agent should avoid the ghosts and successfully clear any board your agent is given. This project will require you to design a useful encoding of your chromosomes, potentially to design a set of high-level/abstract actions for the agent, and to design an appropriate fitness function to guide the learning of your agents.*

*Learning will most likely be more successful with high-level actions rather than only UP/DOWN/RIGHT/LEFT. A sample list of high-level actions is provided below but this list is not exhaustive and you should define your own.*

- *Move to nearest energy pellet ($A^*$ or other search)*
- *Move to nearest extra points piece (like the cherries)*
- *Chase down nearest ghost (if in power mode)*
- *Run away from nearest ghost*

*You should design a fitness function that rewards the agent for performing well and penalizes it for performing badly. You will need to decide what the criteria for performing well and badly are but I suggest that you do not make your fitness function overly complicated. For the fitness function, you should focus on your goals for your agent and not on how the agent will accomplish those goals.*

*Encoding your chromosome will be critical to the success of evolutionary computation methods. For example, as we discussed in class for tic-tac-toe players, you may create a high-level state space and encode a policy of action responses to each state in your individual. Other encodings are quite possible as well.*

*The remaining pieces of an evolutionary computation solution are selection, crossover and mutation. You need to pick approaches to each of these using what we discussed in class.*

*The following is a list of ideas for state space features. Note that this representation lacks a great deal of information. You can't make a general learning agent that uses Pacman's current x,y coordinates given that the boards will vary and you want him to play well in any board. You also can't represent every possible board state for every board in main memory so the problem must become partially observable!*

- *Distance to Inky, Blinky, Pinky, and Clyde*
- *Distance to nearest power pellet*
- *Number of power pellets left*
- *Number of regular pellets left*

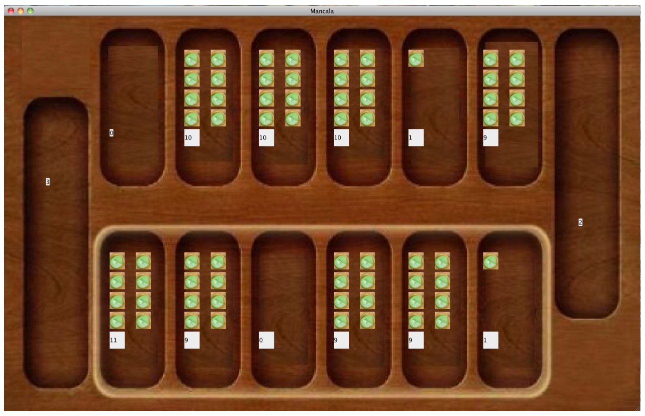


*Figure 5: An example mancala game.*

Although the students focused on controlling Pacman using AI techniques, a viable multi-agent project could be controlling the ghosts intelligently to chase/corner Pacman. The student who wrote the Pacman simulator used it for exactly this purpose, with the goal of creating intelligent teams of ghosts using reinforcement learning.

## Mancala

Although we have described the use of several games for search techniques, students most often arrive in the course wanting to learn about competitive multi-agent games. For this, we need an environment well suited to minimax search. We used the ancient African game of Mancala[6]. Mancala has been used in AI courses at several institutions[7][8][9]. There are a variety of rules for Mancala and we follow the ones described below.

Mancala is a two-player game played on a board with seven "pits" per player. Six of the pits are used as temporary storage for stones during the game. The seventh pit, always the right-most pit from the point of view of the player, is called the mancala and it stores all the stones that a player has won so far during the game. Figure 5 shows the board.

Because mancala is solved for the three, four, five, and six initial stone versions[10], we created a game where the number of initial stones varied randomly between 3 and 8. This number was chosen at the beginning of the game and was available to the agents. This number was used to initialize each of the 12 pits. The game begins with all of the stones evenly distributed across the 12 regular pits. A player can choose to pick up all of the stones from any of the player's six pits. The player then distributes these stones one at a time moving counter-clockwise around the board. A stone is dropped into the player's own mancala when passing it but stones cannot be dropped into the opponent's mancala. The game ends when one player has no stones. At the start

[6]http://en.wikipedia.org/wiki/Mancala
[7]http://www.apl.jhu.edu/ paulmac/ai-prog.html
[8]http://cs.gettysburg.edu/ tneller/cs371/hw6.html
[9]http://www.cs.hmc.edu/courses/2009/spring/cs151/
[10]http://en.wikipedia.org/wiki/Kalah

of the game, the students can choose whether the stones remaining on the opponent's side will be deposited into the opponent's mancala or if they will be ignored. The winner is the player with the most stones. If the extra stones go to the opponent, it is not always an advantage to run out of stones first. Two special rules apply to moving stones. The first is that if a player places the last stone in the mancala, that player receives an extra turn. For example, if there were two stones in the pit two spaces away from the home mancala, the player would place one in the final pit and one in the mancala, gaining an extra turn. We made the second special rule optional, again specified in the initialization file. This rule is that if a player places the final stone in an empty pit on that player's side, that player steals all the opponent's stones from the opposite pit.

Mancala provides an ideal environment for students to explore minimax search, both with and without $\alpha - \beta$ pruning. Using terms from (Russell and Norvig 2009), it is fully observable, strategic, episodic, static, discrete, and multi-agent/competitive. Although the branching factor is low ($b = 6$), students can see the benefits of pruning. In our experience, students often do not fully understand $\alpha - \beta$ pruning until they implement it. In addition, due to the length of the game, students will also explore the creation of effective evaluation functions.

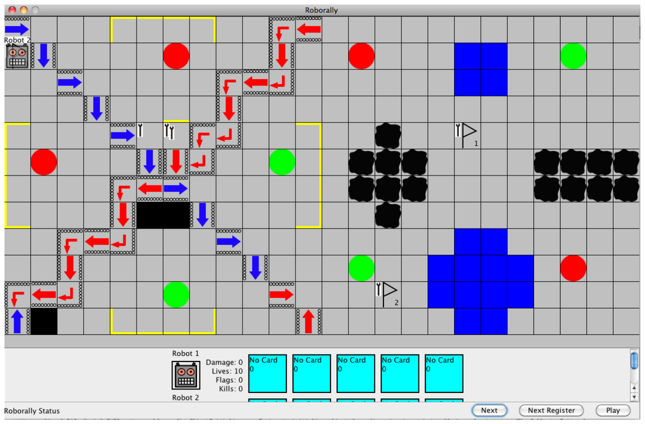
In addition to implementing minimax search with and without alpha-beta pruning, we used Mancala for a project on learning. For this project, both graduate and undergraduate students grew a decision tree to replace their evaluation function for minimax search. Graduate students had to further implement an additional learning method such as regression or clustering. Below is a snippet from this assignment.

*For this project, all students (undergraduate and graduate) will implement decision trees and use them to improve your evaluation functions. There are multiple approaches to using learning to improve your evaluation function and I suggest one here but other approaches are possible! The only rule is that all students must use decision trees and graduate students must also use another learning method from clustering, regression, or kernel regression.*

*One approach to using decision trees in your evaluation function would be to have your agent play hundreds of games against other agents (your own and the heuristics) using the rule variations described above. This agent would record features about each game (e.g. by writing out to a text file) that describe the current game situation and the eventual game outcome. Once sufficient data are acquired, the agent would learn a decision tree to predict the probability of a win at each step and use that tree for the evaluation function. In a similar vein, graduate students could record data about a specific feature and use clustering to discretize that feature and then feed the cluster data into the tree (in addition to the other features). Other approaches are possible!*

## Roborally

We have also implemented and used a game inspired by the Roborally board game, produced by Wizards of the Coast and Hasboro[11]. This is a capture-the-flag type game where robots race to be the first to touch a series of flags while safely navigating an obstacle course. Agents must plan around a variety of interactive board elements and the other agents. Agents interact in a variety of ways including pushing and firing lasers at one another.



Robot 1
Damage: 0
Lives: 10
Flags: 0
Kills: 0
No Card 0
No Card 0
No Card 0
No Card 0
No Card 0
Robot 2
Roborally Status

*Figure 6: Example roborally game with two 12x12 boards next to one another. Within a board, the red and green circles teleport the robot. The blue areas represent water. The yellow lines represent walls, and the arrows show conveyor belts. Black squares are pits, which kill a robot. The robots are currently on the first flag in the upper left corner of the board and they are racing to touch flags 1 and 2 in order.*

The rules are sufficiently complicated that we only briefly describe them here[12]. At each turn, a robot receives a maximum of 9 cards (fewer cards are given to damaged robots). The robot must plan its next five moves (called register cycles). The moves are then executed in sequence with the other robots. The cards enable a robot to move forward one, two or three steps, backwards one step, and to turn right, left or 180 degrees. There are additional option cards that can be used to modify the basic behaviors of the robot and we implemented several of these, enabling a robot to multiply its moves or to better absorb laser fire without being damaged. As the robots move, they can interact in a variety of potentially challenging ways including damaging nearby robots or pushing them to an unintended path. The board elements also interact with the robot by conveying it around the board, rotating it, killing it (if it runs into a pit), damaging it, slowing it down, making it slide, or even enabling the robot to teleport to a far away location. Figures 6 and 7 show examples of roborally games. For each game, a subset of the available board templates is chosen. The flags are placed randomly on blank spaces while ensuring that they are sufficiently far apart.

As with Pacman, we encouraged the students to design board templates and they were quite creative. Several of the student designed boards are highlighted in Figures 6 and 7.

[11]http://en.wikipedia.org/wiki/RoboRally

[12]The entire set of rules is available at http://www.wizards.com/avalonhill/rules/roborally.pdf

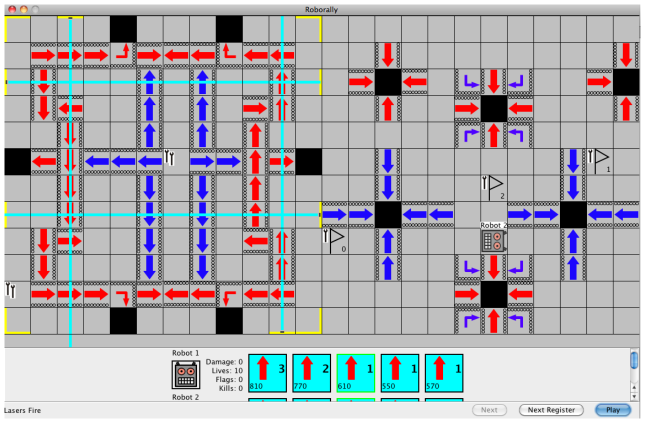


*Figure 7: An example of two other roborally boards, both designed by students. In this case, the game is underway and you can see the lasers firing (blue lines) at the end of the register cycle.*

These boards are challenging and incorporate many of the possible board elements in very interesting ways. For example, the boards in Figure 6 make extensive use of the teleporters. The right hand board in Figure 6 has a large area of water. The boards in Figure 7 contain a variety of conveyors that try to convey the agent into the pit. This makes it tricky for the agent to navigate safely. In addition, the board in the left panel of Figure 7 has several lasers, making it difficult for the agent to avoid damage.

Roborally is an extremely challenging game for an introductory AI class. Using terms from (Russell and Norvig 2009), it is partially observable, stochastic (although the agent's actions are deterministic, the board elements and the other agents provide a stochastic nature to the environment), sequential, static, discrete, and multi-agent/competitive. We used it for projects focusing on A$^*$, genetic algorithms, decision trees, and classical planning. Because the agents do not know what cards they will be dealt in future hands, we introduced rollouts/Monte-Carlo search (Tesauro and Galperin 1996). While this search technique is used in a variety of real-world searches, it was challenging for students to implement.

Below is an example of our project description on learning in this environment.

*The goal for this project is to use learning to make your robot even smarter. Although the goal of touching the most flags in as few steps as possible remains, your score now incorporates kills and deaths (so it behooves you to both stay alive and to kill the other bots):*

$$0.5 * \left(\frac{\textit{total number of kills}}{\textit{total number of deaths}}\right) + 0.5 * \left(\frac{\textit{total number of flags}}{\textit{total number of games}}\right)$$

*Because there are so many ways that you can incorporate learning into your bot's behavior, this project is more open-ended than project 1. If you do want to do something other than the decision-tree approach discussed below, you must talk to us first. Learning methods can be very labor intensive and we want to ensure that you don't choose a project that can't be completed in time.*

*For this project, you should learn at least one probability estimation tree based on the bot's behavior. For example, you could train a tree to predict whether a bot will be successful at hitting an opponent robot. Or you could train a tree to see if your moves will take you to the expected location (e.g. interference from another robot). The possibilities for prediction using trees are quite large! To do this, you will need to write code to save out all possible attributes and data to a data file and then run your agent a number of times to collect data. Once you have the data saved, you should write a decision tree learner that grows a tree based on your data. Once you have the tree trained, you should use the results of the tree back in your bot in an intelligent manner. For example, if you trained a tree to predict probability of hitting another robot, you might choose to be aggressive only if you have a high probability of success.*

## Competition and Creativity

Because many students grew up playing computer/video games, they are motivated and inspired by them (Gee 2007). We encourage this by awarding extra-credit for creative solutions. For example, although the assignment may focus on a search technique such as A$^*$, there is room for creativity in how the technique is implemented within that particular game. Approximately 15-30% of the students were creative enough for extra-credit on their solutions.

For each project, we also run a class-wide competitive ladder. Each ladder runs for 1-2 weeks (depending on the duration of the project) prior to the project's due date. This ladder provides students with a chance to explore a variety of agent behaviors and to see how well they can compete against their fellow students. Since only some students are motivated by competition, the ladder is optional and only used for a limited amount of extra credit. Students must outperform a random agent in order to actually receive the credit. In addition, we enter the ladder ourselves and challenge students to beat us. The top student on the ladder receives one point of extra credit and the second student receives one-half point. A student can receive a maximum of five points of extra credit from the ladder.

## Discussion and Conclusions

We have been teaching AI using a combination of the games described above for six years. We have found these games to be a very effective tool in enabling the class to be a significant learning experience. At the beginning of the projects, numerous students have stated variations of "I understand algorithm X on paper but I can't figure out how to apply it to situation Y." By the end of the projects, they have become experts at the application of these algorithms. They can easily apply the algorithms to new domains and their grades on the exam questions on these algorithms are typically quite high. By the end of the semester, due to the variety and depth of the projects, they have gained considerable confidence in their own skills. They have experience at implementing algorithms in a large existing software product. Additionally, they gain experience in working together on the team-based projects. In our original work with Spacewar, we studied the effects of the games on students' learning and found that the games significantly improved their comprehension on the project topics, as measured by their performance on the exams (McGovern and Fager 2007). We have not yet applied for the necessary Internal Review Board (IRB) approval to study our current students using the wider variety of games presented in this paper although it is planned for future work.

As part of this paper, we are releasing the software for these games. It is available at `http://idea.cs.ou.edu/software/eaai_2011/` As with the original software used for our simulators (McGovern and Fager 2007), this software is provided open source. We also provide copies of all of the assignments used for the games. We would be eager to host multi-institution competitions and interested parties should contact the lead author with such requests.

Since we are actively using this gaming framework in our AI classes, we will continue development. In future work, we would like to add a multi-player game suited for minimax, such as Uno[13] as well as continue to make improvements to the games described above.

## Acknowledgments

The authors wish to acknowledge Jason Fager's original Spacewar framework and all of the students in our AI classes for the past six years who have contributed both ideas and code.

## References

Bayliss, J. D. 2009. Using games in introductory courses: tips from the trenches. In *Proceedings of the ACM SIGCSE technical symposium on Computer Science Education*.

Bloom, B. S. 1956. *Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain*. New York: McKay.

Burns, B. 2008. Teaching the computer science of computer games. *Journal of Computing Sciences in Colleges* 23(3):154–161.

Craig, J. J. 2004. *Introduction to Robotics: Mechanics and Control*. Prentice Hall, third edition.

DeNero, J., and Klein, D. 2010. Teaching introductory artificial intelligence with Pac-man. In *Proceedings of the 1st Symposium on Educational Advances in Artificial Intelligence (EAAI)*.

Felder, R. M., and Brent, R. 2009. Active learning: An introduction. *ASQ Higher Education Brief* 2(4).

Fink, L. D. 2003. *Creating Significant Learning Experiences: An Integrated Approach to Designing College Courses*. Jossey-Bass.

Gee, J. P. 2007. *What video games have to teach us about learning and literacy*. Palgrave Macmillan.

[13]http://en.wikipedia.org/wiki/Uno_(card_game)

Graetz, J. M. 1981. The origin of spacewar. *Creative Computing* 56–67.

Leutenegger, S., and Edgington, J. 2007. A games first approach to teaching introductory programming. In *Proceedings of the ACM SIGCSE technical symposium on Computer Science Education*, 115–118. Academic Press.

McGovern, A., and Fager, J. 2007. Creating significant learning experiences in introductory artificial intelligence. In *Proceedings of SIGCSE 2007, Technical Symposium on Computer Science Education*, 39–43.

Prince, M. 2004. Does active learning work? a review of the research. *Journal of Engineering Education* 93(3):223–231.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition.

Sung, K. 2009. Computer games and traditional CS courses. *Communications of the ACM* 52(12):74–78.

Tesauro, G., and Galperin, G. R. 1996. On-line policy improvement using Monte-Carlo search. In *Advances in Neural Information Processing: Proceedings of the Ninth Conference*. MIT Press.

Woods, D. R.; Felder, R. M.; Rugarcia, A.; and Stice, J. E. 2000. The future of engineering education III developing critical skills. *Chemical Engineering Education* 34(2):108–117.