

Robot Defense: Intelligent Behavior in a Real Time Strategy Game

Introduction:

Computer Games are sophisticated pieces of software and developing a modern computer game requires successfully integrating a variety of core Computer Science skills. As a result, games are a natural vehicle for teaching a range of topics in the CS classroom—all the more so because many students find computer game development to be a motivating and engaging activity.

The primary goal of commercial games is to entertain. As a result, AI techniques are useful only if they improve the game play, or can otherwise make the game more compelling to customers. As a result, the techniques applied in industry are often relatively shallow, providing only enough depth to give the illusion of intelligence. This approach tends to give the greatest returns as resources can then be spent on more visible features (such as art).

At least from a theoretical level, any computer game that generates character behavior (e.g., stimulus response reactions, or path planning) may potentially benefit from the AI techniques that are often discussed in an introductory undergraduate class. Classic board type games like checkers are natural candidates for adversarial search techniques and often form the motivating examples in textbooks. Indeed, many AI programming projects are built around a version of Checkers, Chess, or even Clue. A host of purpose built AI environments also resemble games such as the classic Wumpus World.

Real time games, on the other hand, are less often used in the AI classroom. In part this is probably due to the fact that they are more complicated to implement, and thus they impose a higher burden on the instructor. Nonetheless, real-time games can make very interesting test beds and today form the basis of a variety of academic and military research and simulation environments.

In this project, students build a variety of AI components to control the behavior of non-player characters in a real time strategy game. An initial version of the game is provided: simple functions govern the behavior of non-player characters making the game playable. However, in its initial state, the behavior of various game entities is clearly neither optimal nor general in its nature. Students improve the game incrementally, by implementing AI algorithms. Thus the project described here contains a number of individual assignments. Although the intent is for these assignments to be used together throughout a semester long course, it is also possible for instructors to pick and choose among the assignments, selecting only those that fit the needs of their individual course.

The assignments in this project cover three core areas of Artificial Intelligence and offer a variety of opportunities for discussion. These are described below:

- **Search**
 - o A* search for path planning
 - o Hierarchical A* and A* using simplified graphs
 - o Circumstances when search can be avoided
- **Knowledge Representation**
 - o Procedural vs. declarative representations
 - o Managing state space
 - o Knowledge transparency and explainable AI
 - o Centralization and distribution of knowledge and reasoning
- **Machine Learning**
 - o Reinforcement learning
 - o Operationalizing knowledge through chunking

The Game

Robot Defense takes place in the future, when a strange variety of insects threatens the future of world's food supply. Due only to good fortune, the insects have thus far been isolated to a small island. Scientists must race against time to protect the food supply, before the insects can leave the island. To do this, they must gather samples of the insects, hazardous work since the insects are themselves highly toxic. The player's goal, as the operator of the island's scientific outpost is to gather samples of the insects for scientists to evaluate and to keep them from spreading to the rest of the world.

The game bears similarity to the classic arcade game Lemmings, and the modern flash game Desktop Tower Defense. It takes place on a small, two dimensional map. The initial environment can be modified by the player who can place small structures in a grid-like pattern. Each structure serves to gather insect samples or keep the insects from spreading off the map. Well placed structures serve both purposes.

Robot Defense contains a variety of structures each structure has different attributes, a brief description of each follows:

Vacuum Tower: Vacuum towers capture nearby insects with a powerful suction. The suction is not continuous, but can be turned on and off and directed in any desired direction from the tower. While the tower's suction always produces some force on insects, the magnitude of that force as well as its direction is a function of the local environment's aerodynamics, a function described in more detail below.

Particle Filter: A particle filter provides a barrier for robots and insects alike, although this barrier is fragile and degrades over time. They can be used to direct the insects across the map, but they cannot be used without restraint.

Fan: Unlike Vacuum Towers, fans cannot capture insects. They can however, help to redirect their movement. Thus, fans can be placed to slow down the insect movement or to redirect them towards vacuum towers.

Robot Control Center: The robot control center is the heart of the island operation. Each control center can govern four robots. Robots collect resources from the environment to keep the fans and vacuum towers operating. Resources appear at arbitrary locations on the map over the course of the game. Their appearance can be detected by each control center which must then determine which (if any) robot to assign to the new collection task. Periodically, robots must also carry out tasks specified by mission headquarters. These tasks may require travel to arbitrary locations on the island, and therefore constrain the placement of structures as described in the following section.

Game Play

Initially, the game can be played by a student, and some experimentation with the game is important to help develop an understanding of how the environment works. Play begins with a setup phase in which the map is completely empty. The player places structures of various types onto the map so as to capture as many insects as possible. The placement of structures is limited by two factors. The first is cost: the player has a fixed number of credits to begin with, and each new structure built reduces this value. The second constraint is due to the robots which must periodically leave the map to carry out tasks for mission headquarters. Thus, at all times a path must exist that will allow an arbitrarily selected robot to exit a specified side of the map.

Once structures have been placed on the map, play begins. From this point onward, insects trickle through the map, attempting to cross from the left side to the right (west to east). Insects are simple creatures and will always take the most direct path across the screen (such a path must always exist due to the building constraints mentioned above).

The player engages in the game by performing three types of actions:

- 1) directing robots to acquire resources
- 2) directing the flow of the vacuum towers
- 3) adjusting the direction of the fans

By directing robots to acquire resources from the environment, the player ensures that their structures (fans and vacuums) continue to operate for the duration of the game.

By directing the vacuum towers and adjusting the directions of fans, the players change the aerodynamics of the local environment which in turn affects how well their towers capture insects. That is, the probability that an insect is captured by a particular vacuum tower is a function of the type of insect to which suction is being applied, the location and types of other nearby structures and insects, and the direction and magnitude of other air currents (due to fans or other vacuums operating in the area). The player's task is simply to learn an approximation of this function so they can set vacuum tower and fans to capture a large number of insects. The game is designed so that this function can be set to

one of several known values, or selected randomly. Random selection means that each instance of the game requires the player relearn their strategy.

Assignments

Assignments are designed such that the game's functionality becomes increasingly sophisticated during the course of the semester. The initial game can be played

Project 1: Robot Route Planning. Robots have two tasks that require path planning: gathering resources and performing the periodic off-map tasks assigned by control towers. In this assignment, students implement route finding approaches appropriate to both tasks. Resource gathering requires robots move to arbitrary locations on the map (wherever the resources appear). Search is the best approach to apply to this situation, and students implement A* search using a graph representation of the map provided by our software infrastructure. The second task differs from the first in that the destinations (edges of the map) are static and do not change over time. Because of this, search is not required; instead students can precompute the optimal path from any location on the map without using search and cache this information for fast retrieval later in the game.

Project 2: Rule Based Approaches. After playing the game, students design a simple set of rules to control how robots are assigned to different resource gathering tasks, and to control the settings of fans and vacuums. Rules are built using Soar ([ref](#)), an open source agent architecture which has been in active use since the 1980s. The goal of this assignment is not to build a perfect solution, but rather to distill the students own tactics into a set of well defined rules. This assignment provides a good backdrop for introducing concepts of knowledge representation. We expect that many of the rule sets submitted by students will be relatively brittle, in that they will be designed for one particular aerodynamic function, but will not be robust enough to perform well if they aerodynamic function is selected randomly.

Project 3: Learning a Vacuum Policy. In this project, students implement and use reinforcement learning instead of hand coded rules to identify the value of actions in each particular state. Students compare the results of learning to their hand coded rules for a specific aerodynamic function and then for a set of randomly selected functions. Students are given a preselected series of maps or levels in which the structures have been preallocated and the number and types of insects predefined. Providing standard levels ensures that results are comparable between student submissions and also creates the circumstances in which students can see how increasing environmental complexity impacts the learning algorithm's convergence.

Game Infrastructure

Robot Defense is built using the Java Instructional Gaming (JIG) Project's engine ([ref](#)). Originally proposed in ([ref](#)), the JIG project is a collaborative effort between Washington State University Vancouver and the University of Puget Sound. The JIG Project has three

main aims: to build the software infrastructure to help bring game based projects to students at all levels in the CS curriculum; to create a set of educational resources to foster the use of Java based game projects especially at small, resource-limited, schools; and to develop a community of educators that use and help improve these resources.

The JIG Engine is defined by four key attributes each of which is relevant for the project described in this paper:

Java Implementation There is a consensus in the literature that Java is a leading choice, if not the leading choice, for first programming languages. Moreover, because many institutions focus instruction around a single object-oriented programming language many students will have had more experience with Java than C++ even by the end of their academic career. This is definitely true of the students at our institutions. While C# would be another natural choice for implementation languages especially given the advent of Microsoft's XNA, Java remains a more appropriate choice for our institutions given its prevalent use in our CS curriculum.

2D Focus The authors of the JIG Project have suggested that a focus on 2D gaming helps to make the engine at its resources accessible to students with a wider range of programming skills. In Robot Defense, students will not be interacting with the game engine directly. Rather they will be programming to a specific AI interface. Nonetheless, as researchers from the field of AI, our experience with 3D graphics is somewhat limited, thus the 2D focus provided by the JIG Engine serves dual purposes for our project. First, it makes our task of designing the Robot Defense game simpler. Second, if the AI project does lead students to become interested in game programming, we believe that many students will find it easier to begin their own game design projects with a 2D engine than with a 3D game engine—especially if they have not had a course of computer graphics.