

# Teaching Undergraduate Software Design in a Liberal Arts Environment Using RoboCup

Timothy Huang  
Middlebury College

Mathematics and Computer Science Department  
Middlebury, VT 05753  
1-802-443-2431

huang@middlebury.edu

Frank Swenton  
Middlebury College

Mathematics and Computer Science Department  
Middlebury, VT 05753  
1-802-443-3421

fswenton@middlebury.edu

## ABSTRACT

Most large research universities include a software design or software development course as a required or elective component of an undergraduate computer science major. For various reasons, some institutions, including many liberal arts colleges and primarily undergraduate institutions, do not. In this paper, we present a software design course, tailored to undergraduate computer science students within a liberal arts environment, based on the RoboCup soccer simulation platform. We describe the course curriculum and outline its goals, which student evaluations suggest it achieved. We also outline the features of our “NewKrislet” soccer player, which provides an elementary but sufficiently functional entry point to Robocup client design.

## Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education – *computer science education, curriculum.*

## General Terms

Documentation, Design, Experimentation, Human Factors.

## Keywords

Computer science education, curriculum, software design, RoboCup, multi-agent systems, programming teams, simulation.

## 1. INTRODUCTION

In its December 2001 Final Report, the Joint IEEE Computing Society/ACM Task Force on Computing Curricula proposed four possible approaches to offering core computer science courses at the intermediate undergraduate level, and each approach included a course in software development [5]. Such courses provide students valuable experience with object-oriented analysis and design, software testing and evaluation, API (application programmer interface) programming, software tools and environments, and project management and teamwork.

Most large research universities already include a software design or engineering course as a required or elective component of an

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*ITiCSE'03*, June 30–July 2, 2003, Thessaloniki, Greece.  
Copyright 2003 ACM 1-58113-672-2/03/0006...\$5.00.

undergraduate computer science major. For various reasons, many institutions with a liberal arts emphasis do not. Some institutions would like to offer such a course, but they lack the faculty resources to offer one, perceive that such a course would be too time-consuming for a liberal arts curriculum, or are not familiar with any well-defined, mature platform that facilitates larger-scale projects. Other institutions choose not to offer such a course because they perceive it to be too vocational or lacking in content to fit within a liberal arts environment.

In this paper, we describe an undergraduate software design course based on the RoboCup soccer simulation platform. Several factors make this course well-suited for use at liberal arts or primarily undergraduate institutions. First, RoboCup is an established research and education platform with clear objectives and reliable, readily installed server software. Second, numerous publicly available client software options make it possible to customize courses for a range of time constraints and student backgrounds. Third, a straightforward objective and a set of simple communication protocols make the system easy for both computer science instructors and students to learn. Fourth, the platform encourages discussion of a variety of computer science topics, including real-time programming, multi-agent systems, and artificial intelligence.

Our paper is organized as follows: first, we discuss the RoboCup initiative and some of the ways in which RoboCup has been used for computer science education. Second, we describe our own course and the series of projects assigned to our undergraduate students. Third, we explain the structure and advantages of NewKrislet (NK), the RoboCup client we developed for the course. Finally, we evaluate our success in achieving the course goals based on student feedback.

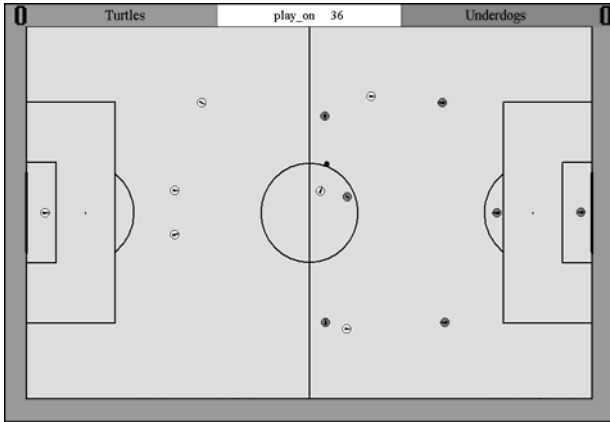
## 2. ROBOCUP

We begin by presenting a brief overview of RoboCup in general and its use as a teaching platform in particular. More comprehensive details can be found in [3]; our aim is simply to provide a sense of the goals, salient features, and pedagogical benefits of the RoboCup platform.

### 2.1 Background

Begun in 1993, RoboCup is an international initiative whose primary goal is to encourage research in artificial intelligence (AI) and robotics by providing a standard environment in which a wide range of technologies and algorithmic approaches can be explored and evaluated. As the name suggests, the standard problem is a soccer match played by robots. International

competitions take place regularly in a software simulation league as well as several physical robot leagues with different size and sensor restrictions. The long-term goal is to develop, by the year 2050, a team of fully autonomous humanoid robots that can defeat the current human champions of the soccer World Cup.



**Figure 1: Snapshot of a match in progress between “Team Turtles” and “Team Underdogs.”**

In the RoboCup software simulation league, two teams of up to 11 simulated autonomous robots play soccer on a virtual field using simplified but fairly realistic models for physics, sensors, communication, and player fatigue (see Figure 1). For any given game, a single soccer server process runs the simulation, generates all sensor information, and communicates with all players. Each player runs as a separate client process that receives sensor information from the server at fixed intervals and generates real-time actions. These clients can run on the same computer as the server or on separate computers. All communication among players takes place via the server using a predefined communications protocol. The structure of the simulation environment provides a challenging domain for research in many areas, including multi-agent systems, uncertain reasoning, real-time decision-making, machine learning, sensor fusion, and software design and management.

## 2.2 RoboCup in Education

Another goal of the RoboCup initiative is to provide a platform for project-oriented educational experiences. With a well-established client-server protocol and a substantial body of publicly available client source code, the RoboCup simulation league provides a rich environment within which undergraduate students can learn about larger-scale software design and analysis, software tools and environments, and teamwork. Furthermore, the well-defined objective and the competitive framework encourage students to develop evaluation methodologies and to measure progress carefully.

To date, however, most formal courses based on RoboCup have been offered at the advanced undergraduate or graduate level and have focused on particular disciplines such as multi-agent systems [9,10], physical robot design [1], artificial intelligence programming [4], and adaptive robotics [8]. The platform itself has been evaluated in the context of its pedagogical utility, though not specifically as the basis for a software design course [6].

## 3. CURRICULUM

Having introduced the RoboCup platform, we now discuss our undergraduate software design course, offered during the winter of 2002. Most of the students had already taken an introductory programming course and a data structures course, both of which used the Java language. Our course met for ten hours per week during an intensive one-month term as part of a “4-month, 1-month, 4-month” academic calendar, though the course could be readily adapted to a quarter-long or semester-long course. We first present our course goals and then describe our assignments.

### 3.1 Course Goals

The primary goals of our course were to provide computer science majors and minors at our institution with opportunities to design and develop larger-scale software projects and to work in a programming team. The secondary goals were to expose students to a wide variety of other software design and computing issues, some more practical and others more theoretical in nature.

On the more practical end, the coursework demanded organization and delegation among members of each programming team, as well as communication about coordinated strategy and tactics for the clients written by different members of the same team. Also, this course gave many students their first exposure to version control, to interacting with an API (application programmer interface), and to maintaining stable versions of client programs for use in competition alongside evolving development versions.

On the more theoretical end, the RoboCup environment encouraged students to address basic issues in artificial intelligence, synchronization and decision-making under uncertainty, and coordination among autonomous clients. Additionally, the client development tasks introduced further challenges, such as access to dynamic data in multithreaded programs, the geometry of local and global polar and Cartesian coordinates, and push-down automata as a method for client control.

We designed a series of assignments, described individually below, to cover both the practical and theoretical issues and to focus on software design issues while addressing relevant computer science topics. We list each assignment as presented to the students, followed by comments on the goals and issues addressed by that assignment.

### 3.2 Assignment #1: Individual Behaviors

**Description:** Each assignment involves modifying and enhancing an existing RoboCup client program. Starting with Krislet, a rudimentary Java-based client [7], design and develop code to accomplish the following behaviors: first, run to the ball and kick it as far as possible. Second, starting from one end of the field, run to the other end and back while minimizing total travel time. Third, starting with the ball at one end of the field, dribble the ball to the other end of the field and back while minimizing total travel time. Work on this assignment individually.

**Comments:** The goal of the first assignment was for students to familiarize themselves with the low-level abilities of individual players and the protocol for interaction between the client and the server. At fixed intervals, the server makes updated sensor information available to each player. The sensor information consists of possibly inaccurate observations about a player’s own

position and velocity as well as observations of objects within the player's field of view. Players can issue real-time, low-level commands to turn, run, kick the ball, and communicate with other players. To accomplish the assigned tasks, students learned how to use and integrate these low-level commands. To minimize the total travel time, students explored how "running" affected a player's stamina level and consequent speed, and they manually tuned their players for optimal performance.

### 3.3 Assignment #2: Cooperative Behaviors

**Description:** Starting with the player from Assignment #1, design and develop two players that pass the ball back and forth, moving it from one end of the field to the other, until one of the players shoots and scores a goal. To prevent uninteresting solutions to this task, each player must touch the ball twice after it has passed midfield. There are no opposing players on the field. Work on this assignment individually.

**Comments:** The goal of this assignment was for students to learn how to coordinate the actions of multiple players. Since each player functions as an autonomous client with no direct access to the state of any other players, each player must sense the locations and actions of teammates in order to accomplish cooperative tasks. Because sensor observations can be noisy, and because they include only those objects in a player's immediate field of view, students learned about reasoning under uncertainty and about the value of maintaining a model of the world state. This assignment gave students the opportunity to experiment with cooperative agent behavior without the presence of adversarial agents.

### 3.4 Assignment #3: Adversarial Challenges

**Description:** Starting with the player from Assignment #2, design a team of two or three client players that scores goals, this time in the presence of an opposing team of Krislet players. Work on this assignment in teams of two.

**Comments:** The goal of this assignment was for students to learn how to accomplish cooperative tasks even in the presence of adversarial agents, i.e., opposing players. The opposing team of Krislet players employed a simple-minded offensive/defensive strategy that could be readily defeated. Nevertheless, Krislet players were skilled enough to steal the ball from a sluggish opponent, so this assignment challenged students to design players that adapt to unexpected events such as intercepted passes and to reason about and experiment with real-time decision-making. It also required that they consider not only offensive but also defensive strategies.

### 3.5 Assignment #4: Soccer Competition

**Description:** Starting with the NK client (an enhanced Krislet client described in Section 4.2), design a seven-player team to compete in an in-class round-robin RoboCup tournament. Work on this in assigned programming teams of four or five students. Document the code and the overall strategy taken by your team.

**Comments:** This was, of course, the most substantial assignment, and students were given the most time to work on it. Among the goals of the assignment were for students to learn to work in programming teams, to specialize the behaviors of the various client players (strikers, defenders, goalie), to coordinate the

strategies of clients with each other, and to develop software testing and evaluation methodologies.

This assignment gave students experience with two of the most significant challenges of the course: planning overall strategy with team members and coordinating the behaviors of clients with different roles. The assignment also taught students the importance of testing code incrementally, utilizing debuggers, and communicating with teammates.

## 4. CLIENT PROGRAM

### 4.1 Client Considerations

With many possible RoboCup clients available for use in our course, we focused on choosing one with a reasonable balance between functionality and learning curve. At one end of the spectrum was Biter [2], which provides a full hierarchy of object-oriented control methods for a client. We decided that the learning curve for this client was too steep for a course aimed at undergraduate students, some with only two prior computer science courses.

At the other end of the spectrum was Krislet [7], a Java-based client that provides a rudimentary interface to the RoboCup server but little to lead toward complex client behaviors. We chose to use Krislet as a starting point and to build upon it for the students, on the principle that the students would be better served by extending a basic client they could fully understand than by trying to merely interface to a substantial base of complex code. This allowed for immediate entry into the process of writing clients for the Robocup platform, requiring minimal startup time for both instructor and student.

For the initial assignments, students were given a simplified Krislet client, and for the last assignments, they were provided with a more functional client, which we now describe in detail.

### 4.2 The NewKrislet Client

To teach students some basic principles of client control and to provide simple examples of their implementation, we modified Krislet by adding several utility and client control features. The resulting client, dubbed NewKrislet or NK, is freely available for download at <http://bj.middlebury.edu/~huang/nkclient.html>. In addition to providing an immediate entry point for the development of Robocup clients, the code illustrates several basic concepts and methods such as multithreading and sharing of dynamic data, local and global coordinates, and push-down automata, inviting the students to explore and refine these aspects of the code to suit their client's needs. We outline below some features of the NK client and their significance to the course.

#### 4.2.1 I/O interface and threaded execution

At its most basic level, the NK client provides an interface to the RoboCup server, allowing connection to the server, receipt of game and client sensory information, and delivery of client commands. One thread of the program is responsible for handling I/O with the server, while another thread has access to the resulting game information, parsed into arrays, strings, and numbers. This threaded approach, which was inherited from the original Krislet client, allows student teams to remain relatively detached from low-level I/O concerns and free to focus on controlling the RoboCup clients. This approach also helps students to better understand multi-threaded program execution,

since the sensory data for each client is updated via a concurrently running thread. The issue of sharing dynamic data across threads is thus present in a restricted context, with methods provided for locking and releasing the current sensory data and for awaiting new data from the server. The implementation of these methods is readily accessible so that more advanced students can work to streamline the client-server communication process as a means of improving client performance.

#### 4.2.2 Local vs. global coordinates

Because all sensory data from the RoboCup server is presented to a client in polar coordinates relative to the player's current position and direction, one of the first technical issues for a client involves determining absolute positions and velocities from this data. To allow focus on higher-level inference and decision-making tasks, the NK client includes rudimentary methods for determining a player's current absolute position and for converting between local (polar) and global (Cartesian) coordinates of observed objects.

The NK client stores all sensory data received from the server in both local and global coordinates. Thus, students can choose which coordinate system to use depending on the requirements of a particular task. The positioning algorithm is quite basic and doesn't explicitly handle noise in the sensor observations. Thus, the NK client supplies immediate rudimentary functionality for the beginning student while inviting the more advanced student to improve the positioning algorithm by implementing, for example, a model of sensor fusion.

#### 4.2.3 Push-down automaton behavior model

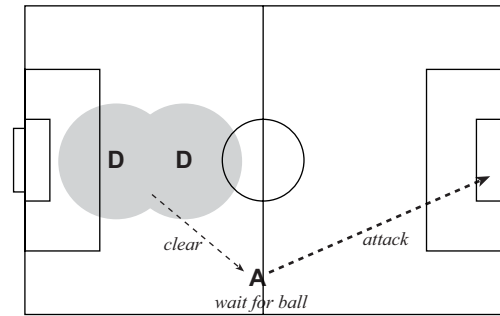
The NK client provides some basic behavior methods via a push-down automaton (PDA) implemented as a finite state machine (whose states are indexed by strings) with a stack of strings for pushing both states and individual RoboCup commands. The NK client provides a few core states, such as one that runs at a certain target, one that kicks at a certain target or position on the field, and one that turns around until a particular target can be seen. These states provide useful alternatives to calling individual RoboCup commands, allowing the user to perform basic tasks in sequence by pushing them onto the stack; after each task has completed, it is popped off the stack to return control to the previous state.

We believe that this model strikes a desirable balance between power and ease of use for the student; the control structure is coded simply as a sequence of if-else clauses for the states, and sequences of commands or states can be issued by pushing strings onto the state stack. This basic implementation allows even a student with little knowledge of such structures to quickly jump in and develop interesting player behavior, without the necessity of an overly steep learning curve. At the same time, it gives more advanced students the flexibility to implement their client's behavior in a more object-oriented manner.

#### 4.2.4 The sample team

Using the PDA architecture, the NK client provides a sample team that solves the course's third assignment of defeating a team of Krislet players. This team's behavior is fairly simple (and entirely reactionary to its chase-and-kick opponent) but provides an example of the use of a PDA to code behavior of a team, from which the students can gain some initial ideas on how states and actions might be used in building cooperative team behavior.

During normal play, the three players of the team operate in two base modes, with one player on *offense* and the other two on *defense*; these states stay at the bottom of the stack throughout play to provide default behavior for the players.



**Figure 2: The NK sample team's basic strategy: defensive players (D) guard specific regions and clear the ball to a waiting attacker (A).**

Within the *offense* state, a player runs to a set position at the midfield line and transitions to the *wait for ball* state, in which it tracks the ball visually. When the ball appears to be within a kickable radius or past the midfield line, the player transitions to the *attack* state, in which it runs toward the ball and kicks it toward the goal. During any point in this process, should the ball cross back into the defensive half of the field, the player transitions back to the base *offense* state.

Each defensive player is assigned a set position and zone of responsibility; in the *defense* state, each player tracks the ball until it enters the player's zone. Once the ball enters its zone, the player runs at the ball and attempts to clear it to a waiting offensive player.

## 5. STUDENT RESPONSE

At the close of the term, we designed and distributed evaluations to the students for course assessment. Over two-thirds of the students agreed or strongly agreed with the statement, "Building RoboCup players has helped improve my programming skills substantially," and nearly 90% agreed or strongly agreed with the statement, "Building our RoboCup players has helped me to improve my ability to communicate and work with others on a software design project." Every student in the course agreed with the statement, "If I had it to do all over again, I would still choose to take the RoboCup course this winter term."

In reaction to the question, "What did you most enjoy about this course?" the written responses fell into several categories:

- competition (9 responses)
- teamwork (4)
- introduction to new concepts in computer science (4)
- seeing their clients' performance (4)—as one student put it, "The elation of getting those little buggers to display semi-intelligent behavior."

In response to the question, "What was the most challenging aspect of building your RoboCup teams?" the students identified:

- teams/cooperation, both of students and of players (7)
- implementation of strategic ideas (3)
- reasoning under uncertainty (2)

We concluded from these responses that, first, the element of competition in our course provided the students with a great deal of extra interest and motivation to excel; at the same time, ample opportunities to observe players behaving in suboptimal and unexpected ways during matches helped keep the competition friendly and light-hearted. Second, the appearance of team-related issues in both categories of responses supports our belief that these matters comprised an integral part of the course experience and that successful human cooperation in the development process was closely tied to successful client cooperation in RoboCup play.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper, we described the curriculum, client software, and benefits of an undergraduate software design course based on the RoboCup software simulation environment. Other courses have utilized RoboCup as a platform to teach topics such as artificial intelligence, multi-agent systems, and robotics, but ours appears to be the first course to utilize it to teach software design to undergraduate students in a liberal arts environment.

The RoboCup software simulation league has proven to be well-suited to our goals. We believe that our assignments and publicly-available NK client software provide a convenient starting point for instructors wishing to utilize the RoboCup platform. The assignments introduce increasingly complex challenges and encourage students to understand a programming interface, to test single-agent and multi-agent performance, to utilize software development and debugging tools, and to work as members of a programming team. The NK client provides a readily understood basic client for students to modify and enhance, and it demonstrates how a client built using a push-down automaton can accomplish complex, high-level tasks. According to course evaluations, most students agreed that the course helped them improve both their software development abilities and their communication and teamwork skills.

Looking forward, we believe that this course can be improved and enhanced in several ways. First, we plan to devote more time to component level design and requirements elicitation. This will give students more insight into how they can work effectively and efficiently on different parts of a team project. Second, we plan to have new students modify clients developed in past course offerings. This will help illustrate the importance of documentation and of exploiting opportunities for software reuse. Third, we plan to conduct design reviews with each programming

team. This will provide students additional experience with team organization, decision-making, and problem resolution.

## 7. REFERENCES

- [1] Birk, A. Autonomous Systems course. 2000. <http://arti.vub.ac.be/~cyrano/COURSES/autosys.html>.
- [2] Buhler, P. and Vidal, J. Biter: A Platform for the Teaching and Research of Multiagent Systems' Design Using RoboCup. In Birk, A., Coradeschi, S., and Tadokoro, S., editors, *RoboCup 2001: Robot Soccer World Cup V. LNCS/LNAI Lecture Notes Volume 2377*. Springer-Verlag, Berlin Heidelberg, 2002.
- [3] Chen, M., et.al. RoboCup Soccer Server Users Manual. 2002. <http://sserver.sourceforge.net/docs/manual.pdf>
- [4] Heintz, F., Kummeneje, J., and Scerri, P. Using Simulated RoboCup to Teach AI in Undergraduate Education. In *Proceedings of the 7<sup>th</sup> Scandinavian Conference on AI*. (Odense, Denmark, Feb., 2001).
- [5] Joint IEEE Computing Society/ACM Task Force on Computing Curricula. *Computing Curricula 2001 Final Report*. (Dec. 2001), <http://www.computer.org/education/cc2001/final/index.htm>.
- [6] Kummeneje, J. RoboCup as a Means to Research, Education, and Dissemination, Licentiate of Philosophy thesis. 2001. Department of Computer and Systems Sciences, Stockholm University and the Royal Institute of Technology. <http://www.dsv.su.se/~johank/publications/2001/licentiatethesis.pdf>
- [7] Langner, K. Krislet RoboCup client. <http://www.ida.liu.se/~frehe/RoboCup/Libs/Sources/krislet-0.1.tar.gz>
- [8] Lund, H. H. Adaptive Robots course. 1999. [http://www.daimi.au.dk/~hhl/adap\\_rob99.html](http://www.daimi.au.dk/~hhl/adap_rob99.html).
- [9] Malec, J. CD5320: RoboCup course. 2000. <http://www.idt.mdh.se/kurser/cd5320/>.
- [10] Vidal, J. and Buhler, P. Using RoboCup to Teach Multiagent Systems and the Distributed Mindset. In *Proceedings of the 33<sup>rd</sup> ACM Technical Symposium on Computer Science Education*. (Covington KY, Feb. 2002).