Abstract

This project builds upon the Robocup soccer simulation league, using many parts of this system to implement a combined 3-dimensional graphical interface and full 22-player client system. Robocup consists of three primary systems: the server, which maintains the current world state; the client, which represents a player, and receives data from the server, processes this data, and sends action commands back to the server; and the monitor, which receives perfect world information from the server and then renders the scene graphically. Our project concentrates on artificial intelligence and machine learning, incorporating elements from genetic algorithms, memetic learning, and traditional artificial intelligence. We use these simple traditional methods to attempt to implement both simple behavioral responses (passing, dribbling, shooting) and rudimentary teamwork strategies. For the goalie, however, we use both a genetic and memetic algorithm to learn goalie ball blocking. We also study in the efficacy of each learning method in this environment. To accomplish these tasks, we designed a system capable of communicating with the Robocup server as a monitor, coach, and team of clients. Data concerning the state of the field is received from the server, parsed, and stored. User input is then processed (the user can interactively control one client), and the client AI is computed. The clients then send commands to the server, and the scene is drawn from a variety of selectable cameras. To complete the project, we integrated these features and additional elements from both the client and monitor into one Windows executable.

Keywords: Robocup, Artificial Intelligence, Genetic / Memetic Algorithms, 3D Graphics
I. Introduction

Robocup, an ongoing international project related to artificial intelligence and robotics, desires to, “by the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.” [1] While the achievement of this goal lies some distance in the future, a competition is held each year pitting design teams against each other in a robotic soccer tournament. Several categories exist in the Robocup challenge, and the one most relevant to our work is the simulation division, which places two software AI soccer teams into a simulated environment and allows them to compete in a mock soccer game. For our project, we use genetic and memetic algorithms to create an adaptive learning soccer AI agent which is integrated with basic client AI and a separately developed 3-dimensional graphics engine.

The soccer simulator consists of three main components:

1. **Server** – This controls the state of the world, and all of the simulation aspects of the program are taken care of by the server. The server returns play state, visual and auditory information to the client as well as information about the client itself (such as remaining stamina, etc.). It also sends out complete data as to the state of the field to the monitor module.

2. **Client** – This controls the “mental state” of the player. The client essentially receives appropriate visual and auditory data from the server, and then, using this data and the players “mind” and “memory,” makes a decision what to do at the particular time step and relays this information to the server. The client performs actions by sending basic action commands to the server.

3. **Graphics Engine** – This, developed in conjunction with the CS 490 project, renders the game world to the screen and is responsible for handling inputted user control. The graphics engine communicates directly with the server, receiving full information concerning the state of the world.

The program operates in two modes:

1. **Player vs. computer** – Here the human player will control a single player on a team, while the remaining 21 players will be computer controlled.

2. **Computer vs. Computer** – This mode places the human player in the role of observer as two fully autonomous teams duke it out on the field.

Rob was primarily concerned with developing the adaptive learning aspects of the autonomous agents, while I worked a bit on traditional AI strategies, but mainly the larger system – communication, message parsing, environmental representation, learning implementation, control, camera modes, etc. Of course, due to the nature of the project, we worked more as a team than individuals, thus overlap was common.
II: Related Research

The Robocup soccer simulation league consists of three primary components: the server, the client, and the monitor. Of these, we did not alter the server, but we created and combined the client and monitor. Much of the following information, and substantially more, can be found in the Robocup Soccer Server User Manual. [6]

1. **Server Module:** The Robocup server is the *supreme commander* of the soccer simulation world in which we worked. It both sends and receives messages to the client and monitor modules. Clients connect to the server using UDP/IP sockets (port 6000), with each connection representing one player. The server is responsible for keeping track of the positions of each player, the ball position, the current game state (kick off, corner kick, foul, etc), the physical attributes of each player (stamina, movement constraints, field of view, etc) and the ball (velocity vector, acceleration, etc), the game clock, teams and player numbers, referees, and various other state-of-the-game elements. The server operations in discrete intervals, sending and receiving data every 100 ms.

2. **Monitor Module:** The monitor module accepts data structures containing perfect information about the field from the server and graphically displays these elements. Like the client, the monitor connects using UDP/IP sockets (also port 6000). The monitor can both send messages (say, to signal a kickoff) and receive messages from the server. Monitor connections come in two flavors, version 1, which we used, and the more robust (but substantially more complicated) version 2. At each time interval, the server sends a dispinfo_t structure to each connected monitor. This structure contains the following information:

   a. A structure containing the currently active game play mode, the two team names, the score, the game time, and an array of player position structures.

   b. Each player position structure contains the player’s team, the player’s uniform number, the player’s $x$ and $y$ position, and the angle the player is currently facing.

   c. A structure containing the messages from the players and the referee.

   d. A structure containing drawing information.

The monitor is responsible for receiving, parsing, and graphically displaying this information.

3. **Client Module:** Each client represents only one player. After a connection with the server is established, the client can send and receive plain text messages to and from the server. Clients are responsible for initializing each player to a team, placing him on the field of play, and, following the kickoff, receiving server messages, parsing these, formulating a strategy, and replying to the server with a message detailing the action the player wishes to perform during the current time interval. Information about the game state the client receives from the server is
imperfect – the server realistically models both sight and sound, so any particular client is never fully aware of the entire field of play.

4. **Trainer/Coach**: In addition to the client and monitor, both a trainer and a coach can connect to the server (although the trainer uses port 6001). A coach receives perfect information about the field, like the monitor, but additionally can broadcast messages to the players. The trainer, who also receives perfect field information, is essentially omniscient – he can change the current game state, move players to any location, and position the ball at any given point and velocity. This is useful for testing and automated learning techniques.

### III: The Graphics Engine / Client Class Objects

At the base level, our graphics interface is built upon the “modeler” project framework supplied for computer graphics courses at Yale, MIT, and the University of Washington. [2] The framework utilizes the Faster Than Light Toolkit (FLTK) cross-platform windowing system. [3] A number of global class objects are responsible for the various systems and objects:

1. **Object Class**: Used to represent all the objects which will be drawn, including players, the ball, the goals, etc. This class contains a number of core attribute variables, such as the current position of the object, the previous object position, and the object’s movement vector, scale, and rotation. Functions are provided to increment or update each of these values.

2. **Geometry Class**: A basic class containing attributes for the individual geometric primitives of each object. Physical attributes like the radius, height, position, scale, rotation, etc. for each object are stored and easily modified, making this class quite useful for creating simple interpolated keyframe animation.

3. **Ball Class**: Contains functions to draw the soccer ball.

4. **Goal Class**: Contains functions to draw the goals.

5. **Music Class**: Contains functions which implement DirectX 8 music. [4]

6. **Sound Class**: Contains functions which implement 3-D DirectX 8 sound effects. [4]

7. **Replay Class**: Contains functions to record and play back, at a variety of speeds, the action occurring on the field. Basically, at each time interval, this class stores the locations and rotation of each visible object, and then cycles through this array to replay the action.

8. **Net Class**: Contains high level communication functions. These provide functionality so that the monitor, coach, and clients can create UDP/IP sockets with the server and both send and receive messages. This class contains structures which store the established socket connections for the monitor and clients. The lower-level UPD/IP functionality is mainly borrowed form the TsingHua Robocup Team. [11]

9. **Parse Class**: Contains the data structures used to parse the monitor data sent via the net class.
10. **Coach Class:** Contains trainer functionality. This allows us to place the ball and players on the field at will – essential functionality for autonomous learning. Includes functions to place the ball at a given location with a given velocity, place a player at a given location and rotation, and to alter the game play state on the fly.

11. **Player Class:** Contains functions to draw the players, as well as a number of variables and functions to manipulate the player objects themselves. Keeps track of the player’s team, uniform number, happiness; the distance, angle, and number of both closest teammate and closest opponent to the player; the boundaries of the field area the player is assigned; and the coordinates for the center of this area.

12. **AI Class:** Contains functions to execute the player artificial intelligence. Contains structures to keep track of a variety of situational data, such as the distance and angle from each player to every other player, the closest player to the ball for each team, and the distance and angle of each player to the ball. Functions are provided to calculate the above information, and to dribble, pass, shoot, run to a location, turn to a specific heading, dash, sprint to the ball, calculate player happiness, and decide upon and execute a strategy for each player.

13. **Goalie Class:** Contains a number of variables and functions at a child level for use in the genetic and memetic learning algorithms. Keeps track of the fitness for each child, the results of each trial, the angles to be tested, the fitness rank, and the eligibility and confidence for each angle. Includes functions to decay the confidence, evaluate each child, sum the confidence, randomize the starting values, determine fitness, add eligibility, and mutate each gene.

14. **Genetic Algorithm Class:** Contains variables and functions to control the genetic and memetic algorithms, including arrays of Goalies for the children, parents, elites, and super gene. Includes functions to find the best children, select a child to become a parent, initialize each child, and perform mutation and crossover.

**IV: Flow of the Game**

**Initialization:** Before loading the client, the server must be running on the ‘localhost’ machine. Upon execution, the client first runs through a brief initialization routine, which prepares the windows, sets up the sound and music systems, loads the application textures, sets the player properties, and initializes the network. The 22 players are given field area boundaries, inside of which they are happy, but as they approach the edges, they become less satisfied with their current position. The center of these areas is also defined, and the location of the opposing goal is set.

The network initialization consists of first initializing the client and monitor by creating a UDP/IP socket to the server and sending a connection message. The team names are set, and each player establishes a socket with the server and sends an initialization message identifying its position (if the goalie) and its team name. The connection sockets are stored in a global Net object.
The final part of the initialization procedure is to place each player in his proper position on the field. Before the kickoff, each player can send a ‘(move x y)’ command to instantly teleport to the (x, y) point. After the game has begun, this functionality is disabled (although the trainer/coach still possesses this ability).

**Control:** Once we enter the main game loop, we first check for user input. The user can control at most one player at any given time, although the game can also run completely autonomously. When a player is selected, the artificial intelligence for that player is disabled, and the user can move the player forward and backward, turn left and right, kick the ball lightly forward or to the left or right (dribbling), kick the ball moderately hard (passing), or shoot the ball. Since Robocup does not implement a ‘possession’ feature, all the ball movement functions are accomplished by kicking the ball at varying speeds and in varying directions. The player can also control the replay feature – pausing, rewinding, playing, and fast-forwarding the recorded loop. Controls are also provided to switch between any player on Team 1 (except the goalie), switch or select a camera mode, and play or stop the music loop.

**Receive and Parse Monitor Message:** The next task is to receive and parse the information contained in the dispinfo_t structure sent from the server to the monitor at each time interval. Once we have used our UPD/IP socket to receive the message, we examine the contents of the showinfo_t structure contained within, which contains the vital information about the object positions and game state. We store the play mode, the time, and the score. Next, we process the coordinates for the ball, storing its position in the global Ball object. We repeat this process for each player, storing the team name, uniform number, position, and direction for each of the 22 players. We then compare these updated values with those from the previous time interval to calculate movement vectors for the players and ball.

**Player Update:** Once we have the full field information, we process the artificial intelligence for each player. This can occur in two ways. First, we can be in the process of training the goalie AI using memetic or genetic algorithms. Or second, the regular player AI can operate within the normal routine of the game.

1. **Player AI:** The player AI is hard-coded into the client and obeys a very narrow range of rules. First, if the game has not yet begun, each player is in the state awaiting kickoff. One player from the kicking team will approach the ball and pass it to his teammate. Once the game has begun, each team will have one active player whom is closest to the ball. This player enters a defensive or offensive state depending on which part of the field he is in at the time. If the player is in the defensive third of the field, he will pursue the ball and attempt to kick it forward, towards the opposing team’s goal, clearing the immediate threat. If a player is active towards the offensive end of the field, he will dribble at the goal, and, once within a reasonable range, shoot the ball at the goal. Further, each player is given a boundary which he does not like to cross, so when an active player reaches this boundary, he will attempt to pass the ball to the closest teammate. When a player is not active, he enters a state during which he seeks to
get into an open position where he can receive a pass. Each player is given a happiness level, which is affected by proximity to both opposing and friendly players, as well as the ball, and the distance he has traveled towards his boundary position. Whenever a player’s happiness drops to the point at which he exceeds a given threshold, he attempts to travel in a direction which makes him happier.

2. **Goalie Learning (Genetic Algorithm):** We use the trainer to autonomously train our goalie. Essentially, the trainer kicks the ball at the goalie, and the goalie performs a series of turns and dashes according to the particular set of angles that comprise his genes. Depending upon where the ball is relative to his position at a given time, he chooses an angle from this stored array, turns in that direction, and dashes. We are evolving the angles in this stored array, such that the goalie will always turn in the right direction for a given shot approaching the goal at a given vector and angle. If the goalie manages to successfully intercept and catch the ball, or if the shot misses wide, his fitness is increased. This is repeated for some number of tests for each child in the generation. Selection is the process whereby two children are selected to become the parents of the following generation. We use a roulette-wheel based selection function, where each child is given a weight proportional to his fitness and that of his siblings, and then a winner is chosen. This helps prevent reaching a state of local minima if we always select the children with the highest fitness. Further, a given number of children can pass from one generation to the next untouched; this process is know as elitism. Once parents have been selected, the crossover function randomly selects a parent and sets two genes, randomly, to that of the other parent. The attributes of the child, after crossover, then undergo a process of mutation, which also adds a touch of randomness to the algorithm so that local minima and maxima are avoided. Our mutation function works by altering each attribute by +/- some randomly selected value within a given range. So, if the range were 3 to -3, for example, each turn angle would be mutated by some random number between 3 and -3. Finally, these new children are inserted into the next population, and the process begins anew.

3. **Goalie Learning (Memetic Algorithm):** The memetic approach to goalie learning is similar to that implemented in the genetic algorithm, except that it operates on each gene attribute, rather than each child. The idea behind memetic algorithms is to place additional emphasis on those decisions likely to have had the most impact on the outcome of the event. Thus, each attribute (the array of angles, in this case) is given an eligibility based on how likely it is to affect the outcome. In our system, this eligibility decays based upon time, so that the last turns and dashes made are given the most importance in determining the outcome. The confidence for each attribute is calculated by multiplying the reward (given for successful blocks or shots which travel wide of the goal) by the eligibility. Those attributes with the most confidence are selected to carry over into the next generation. A similar process of selection, crossover, and mutation to that in the genetic algorithm occurs, although selection can occur by selecting a child based on overall confidence, or by selecting individual genes based on their
confidences. A ‘super gene’ is also passed on to the next generation, which is essentially a child with each gene having the highest confidence of the previous generation. Again, like in genetic algorithms, the children are then inserted into the new population, and the process repeats.

**Drawing the World:** After we have gathered and parsed all the server information about the world, and once we have updated the player information and user input, we draw the world to the screen. This is relatively easy, since we store the world information as part of the actual objects which are drawn to the screen. Each visible object contains a draw function, which is called at this point. This function contains all the nitty-gritty details of drawing the actual primitives, supplying color and texture information, etc. Since we do not support the importation of external model file formats, the geometry for each model is hard-coded into the program.

**Extra Features:** We have implemented a number of camera modes to increase the immersive qualities of the monitor. These can be switched on the fly, although they are locked in place, thus the mouse cannot be used to alter the viewing angle, position, or zoom. The seven camera modes include:

1. **Default:** Seen from the side with the camera located slightly above the field and located at midfield, this view gives a nice perspective look at the entire field, as if viewed by a spectator from the bleachers.
2. **Top Down, Tracks Player:** This camera is located directly above the currently selected player, tracking his movements as he runs around the field. Approximately one-half of the field is visible at any given moment.
3. **Over-the-shoulder Chase Cam, Tracks Player:** This camera is located behind and slightly above the selected player, rotating and moving so that it remains directly behind the player and facing the same direction as the player.
4. **Floating, Tracks Ball:** This camera floats above the field, rotating in a pleasing manner as it tracks the ball being kicked around the field.
5. **Top Down, Full Field:** Located directly above the field, this view faces straight down, with the entire field visible at all times. Does not rotate or track.
6. **First Person:** This camera places the viewer into the head of the currently selected player. The view faces straight ahead in the direction of the player, and rotates and moves as he does.
7. **Top Down, Tracks Ball:** Identical to the top down, player tracking camera, expect that it tracks the ball instead of the player.

We have implemented simple music and sound effects using DirectX 8, although the code comes mainly from the book *OpenGL Game Programming* by Hawkins, Astle, and LaMothe. We slightly modified the functions supplied in their tutorials and stripped much of the Windows code, but the core functions remain essentially unchanged. However, sound effects and music can now be easily added to the game. We include only one sound effect which plays when the user kicks the ball, and a game loop which includes a medley of soccer anthems.
V: Results

Given the magnitude of the system we created, we were quite pleasantly surprised with the functionality of the final product. Communication, message parsing, environmental representation, and the actual drawing of the scene all work well, and the artificial intelligence and learning environment we implemented is at the least very functional, even if the AI is not spectacular.

The hard-coded player AI is very buggy and unreliable, although it can at times be quite humorous to watch, especially for those of us that remember our days in the Under-8 youth soccer division. Due to the emphasis we placed on implementing goalie learning, the field player AI suffered simply due to time constraints.

The learning, however, was at the same time successful and somewhat disappointing. While we were able to obtain satisfactory – even pleasant – results using the genetic algorithm, the memetic algorithm was unreliable at best. The genetic algorithm performed admirably in many of our trials, with the goalie becoming noticeably more reliable in catching the ball with each successive generation. The results from the genetic algorithm are plotted in Figures 1 and 2.

![Figure 1: Results from the genetic algorithm, charting the average fitness of each generation. These results were generated using 20 children per generation, and 40 tests per child. Fitness is determined using the function: f = 0.25 * Misses wide – 0.25 * Goals scored + 3.0 * Balls blocked](image)
VI: Future Work

The system we have constructed is well suited for future work in both the areas of the Robocup simulation league and 3-dimensional graphics. Within the framework of Robocup, the artificial intelligence can always get better, especially the hard-coded player AI, which we somewhat neglected, instead concentrating on the goalie learning. As far as the client goes, the AI is very open-ended, and relatively easy to manipulate within the framework we have constructed. The memetic algorithm, in particular, could stand to be revised. Also, currently the learning is handled separately from the game. It would be a useful addition to merge the learning and the game, such that the goalie could be trained before the game, and then would utilize that training during the game.

As for the graphics and interface, a number of improvements spring to mind. A visual on-screen display showing the team names, score, time, and other items would be nice. The ability to use the mouse to zoom, pan, and rotate the camera in any of the predefined settings would be useful. The camera model as a whole could be spruced up quite a bit. One might want to implement a version 2 monitor system, which would allow the user to interactively control a number of attributes, such as the ball position. User control could be greatly improved, as could client dribbling, passing, shooting, etc. The music and sound classes could be improved, as they currently are essentially direct ports from the tutorials in OpenGL Game Programming. Finally, a more accurate visual collision detection system could be implemented.
VII: References