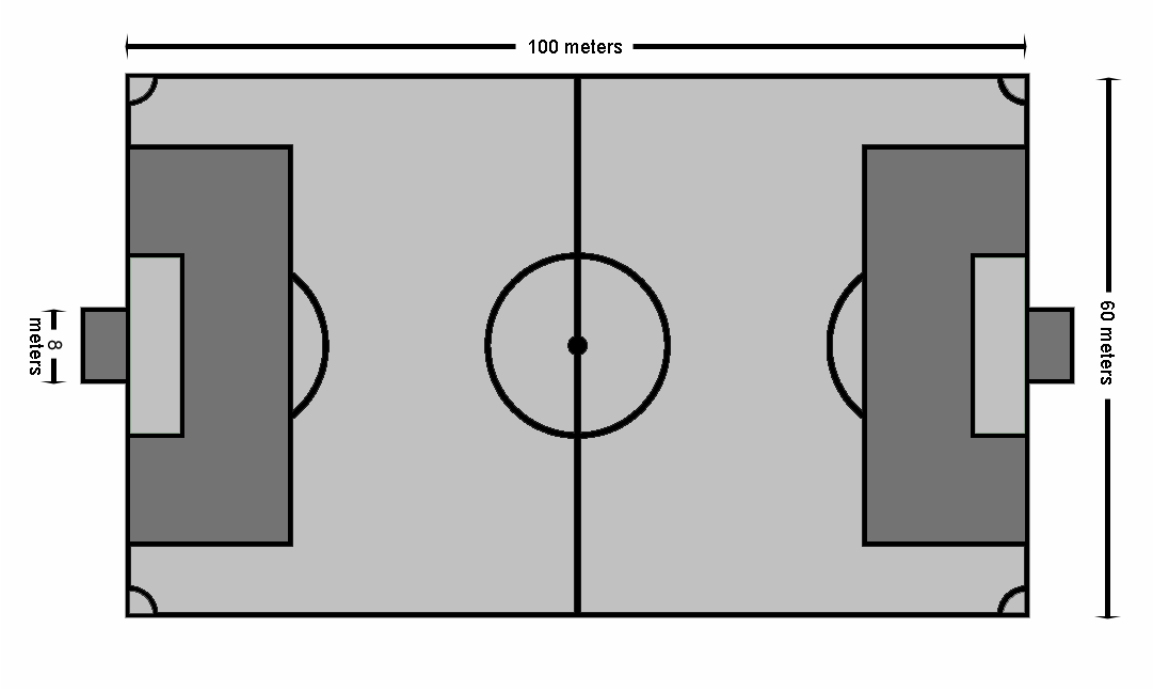
UIST 2003 Interface Design Competition
User Guide

Dan Maynes-Aminzade
Last Updated: August 8, 2003

Field Layout

The soccer field is 100 meters long and 60 meters wide. Goal areas are 8 meters wide, centered on each side of the field. In specifying the coordinates of objects on the soccer field, the server uses an absolute coordinate system in which the top left corner of the field is (0,0) and the bottom right is (100,60).



Player Control

Each contestant has control of five players. By sending messages to the server, contestants can change the instantaneous velocity of each player and send each player a kick command. The maximum player velocity allowed by the server is three meters per second and the minimum time interval between kicks is 0.25 seconds.

Kick Model

In order for a kick to be successful, the kicking player must be no farther than 2 meters from the ball. Kick strength is specified as a floating point value ranging from 0.0 to 5.0. A successful kick imparts a velocity of to the ball with the same magnitude (in

meters/second) as the kick strength in the direction of the vector from player's position to the ball's position. This velocity is added to the ball's velocity before the kick. If multiple successful kicks occur in a single frame, the effect of these kicks is additive.

Ball Model

The ball starts in the center of the field (50,30), and remains there until it is kicked by one of the players. Once the ball is in motion, its velocity attenuates by a factor of 0.98 each frame. The ball cannot go out of bounds; instead, it collides and rebounds with the edges of the field. This collision is perfectly elastic and does not decrease the ball's speed.

Message Format

All messages to and from the server use the `DATAGRAM` structure, composed of 4-byte integers and 4-byte floats in network byte order. The `DATAGRAM` structure is specified as follows:

```
#pragma pack( push, 1 )

typedef struct Player {
      float x,y;
      float vx,vy;
      float kick;
} PLAYER;
typedef struct Ball {
      float x,y;
      float vx,vy;
} BALL;
typedef struct Datagram {
      long team;
      long paused;
      long score[2];
      BALL ball;
      PLAYER team0[5];
      PLAYER team1[5];
} DATAGRAM;

#pragma pack( pop )
```

If you are composing a datagram to send to the server, you only need to fill in a few fields:

- the `team` field (0 or 1) specifies your team number
- the `team0` or `team1` field, depending on your team number:
    - within the `team0`/`team1` field, the `vx` and `vy` specify the velocity of each player
    - within the `team0`/`team1` field, the `kick` value specifies kick strength for each player executing a kick

All other fields are ignored by the server.

If you are receiving a message from the server, all of the fields in the `DATAGRAM` structure are filled in. You can use these values to update your local copy of the game state. This is useful if you'd like to create your own game visualization.

Byte Ordering

If you are building a datagram in Java, you will not need to worry about byte ordering, since Java I/O uses network byte ordering by default. Simply open a `DataOutputStream` and use `writeInt` and `writeFloat` methods to build up a datagram. You can find an example of this in the file `Datagram.java` in the reference implementation.

If you are building a datagram in C/C++, you will need to convert values from host byte order to network byte order before sending them to the server. Likewise, values received from the server must be converted from network byte order to host byte order. To convert 4-byte integers, use the built-in compiler macros htonl() and ntohl(). To convert 4-byte floats, you can use the routines htonf() and ntohf() provided in the reference implementation. See the file `precomp.h` in the Windows client implementation and `soccerclient.h` in the Unix client implementation.

Connecting to the Server

The game server listens for UDP datagrams on port 25618. When it receives a datagram from a new IP address, it begins broadcasting status update datagrams to this address on port 25619.

Only one client is allowed to connect per team. Once a client has connected by sending a datagram specifying its team number, other clients attempting to connect using the same team number will be ignored. Messages from the same client specifying a different team number will also be ignored. In other words, your client cannot change teams in the middle of a game; the server must be restarted first.

The server state is updated at intervals of 60 milliseconds. Please do not send messages to the server at a rate faster than one datagram every 60 milliseconds. Sending multiple datagrams within a single server frame offers no advantage and will only bog down the server. Clients that flood the server with excessive data packets may be disqualified from the competition.

Getting Started

The game server runs only on the Windows platform. To start the server, simply run `SoccerServer.exe` in the `soccerserver` folder. The visualization requires a reasonably fast graphics card; if you experience a low frame rate, try making the window smaller. When the server starts, the game is paused; to pause and unpause the game, simply click anywhere in the window.

Once the server is running, compile and build one of the sample clients. There are three sample clients available, one written in Java, one written in C for the Unix platform, and one written in C++ for the Windows platform. The following sections include instructions for compiling and running each sample client.

Windows Client

The source for the Windows client is in the `windows_client` folder. The folder contains two Microsoft Visual Studio project files: soccerclient.dsw is for Visual Studio 6.0, and soccerclient.sln is for Visual Studio 7.0. Open the appropriate project file for your version of Visual Studio and build the project.

Once the executable is built, run it from the command line. There is only one command line argument, the IP address of the computer running the game server. If you are running the client on the same machine as the server, you can specify 127.0.0.1 (localhost) as the IP address, and run the client with the command:

```
soccerclient.exe 127.0.0.1
```

The client displays instructions for controlling the players using the keyboard.

Java Client

The source for the Java client is in the `java_client` folder. To build the client, change to this folder and run the command:

```
javac *.java
```

Once the class files are built, run the client from the command line. There is only one command line argument, the hostname of the computer running the game server. If you are running the client on the same machine as the server, you can specify localhost as the hostname, and run the client with the command:

```
java SoccerClient localhost
```

The client displays instructions for controlling the players using the keyboard.

Unix Client

The source for the Unix client is in the `unix_client` folder. To build the client, change to this folder and run the command:

```
make
```

Once the executable has compiled, run it from the command line. There is only one command line argument, the hostname of the computer running the game server. If you

are running the client on the same machine as the server, you can specify localhost as the hostname, and run the client with the command:

```
./soccerclient localhost
```

The client displays instructions for controlling the players using the keyboard.