Introduction to RoboCup

Michael Floyd

March 1, 2012

Outline

- Overview of RoboCup
- How a simulated game is played
- Client-Server communications

Overview

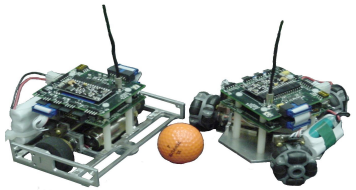
- "*By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team*."

- A standard problem for AI research

- Workshops, conferences and yearly competitions

Soccer Leagues

four-legged



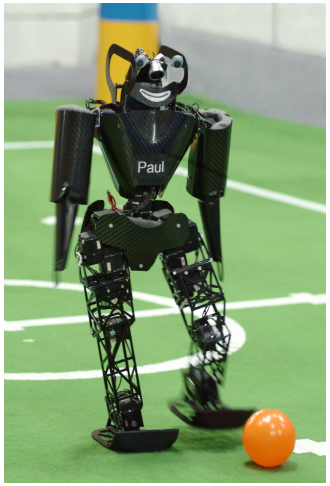
small size

middle size

simulation

Soccer Leagues (2)

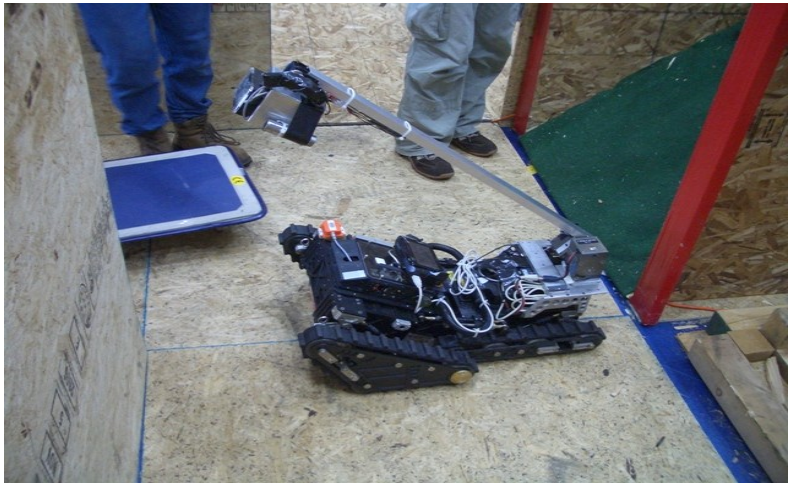
Humanoid:



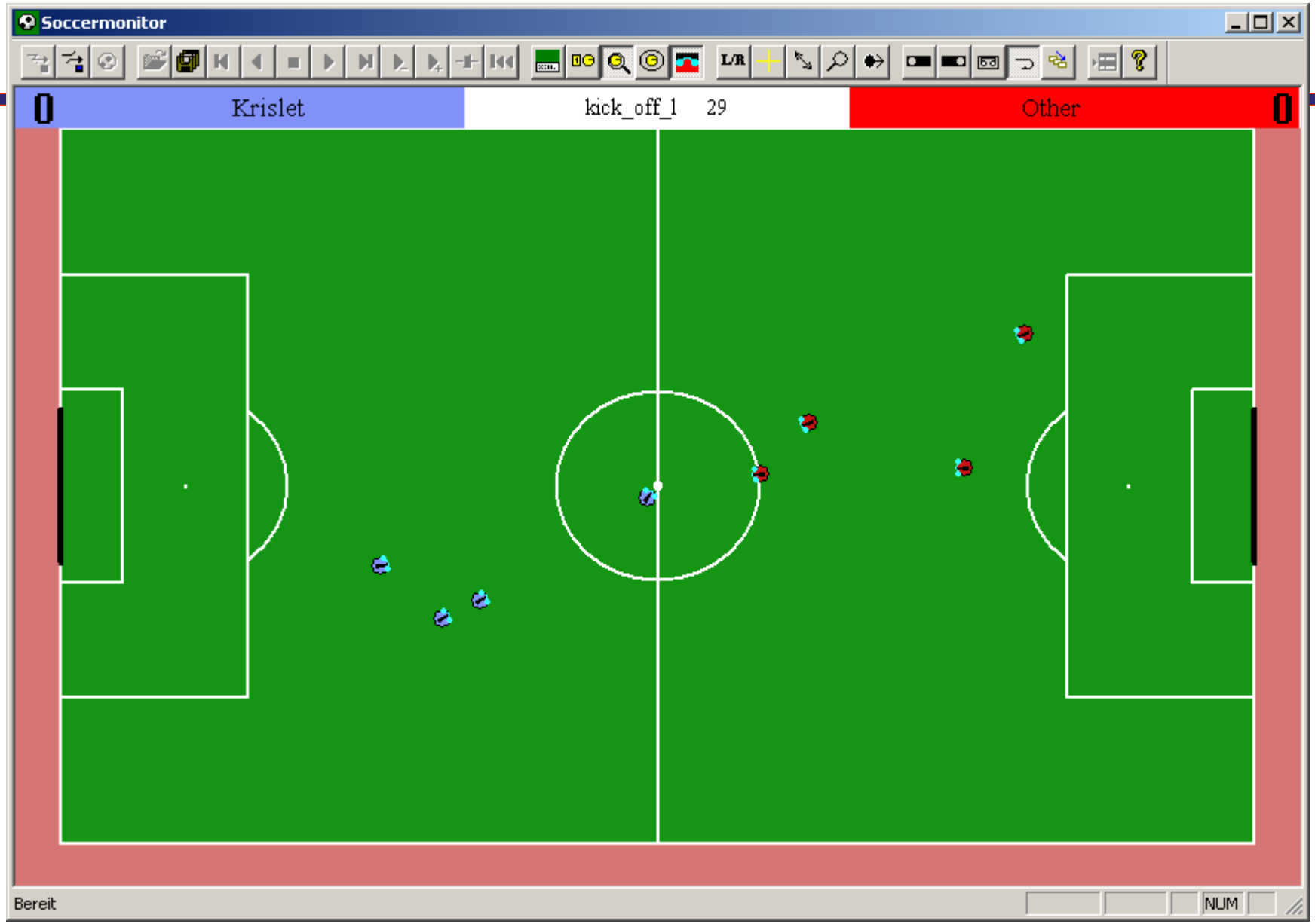
Stardard Platform (Nao):

Other Competitions

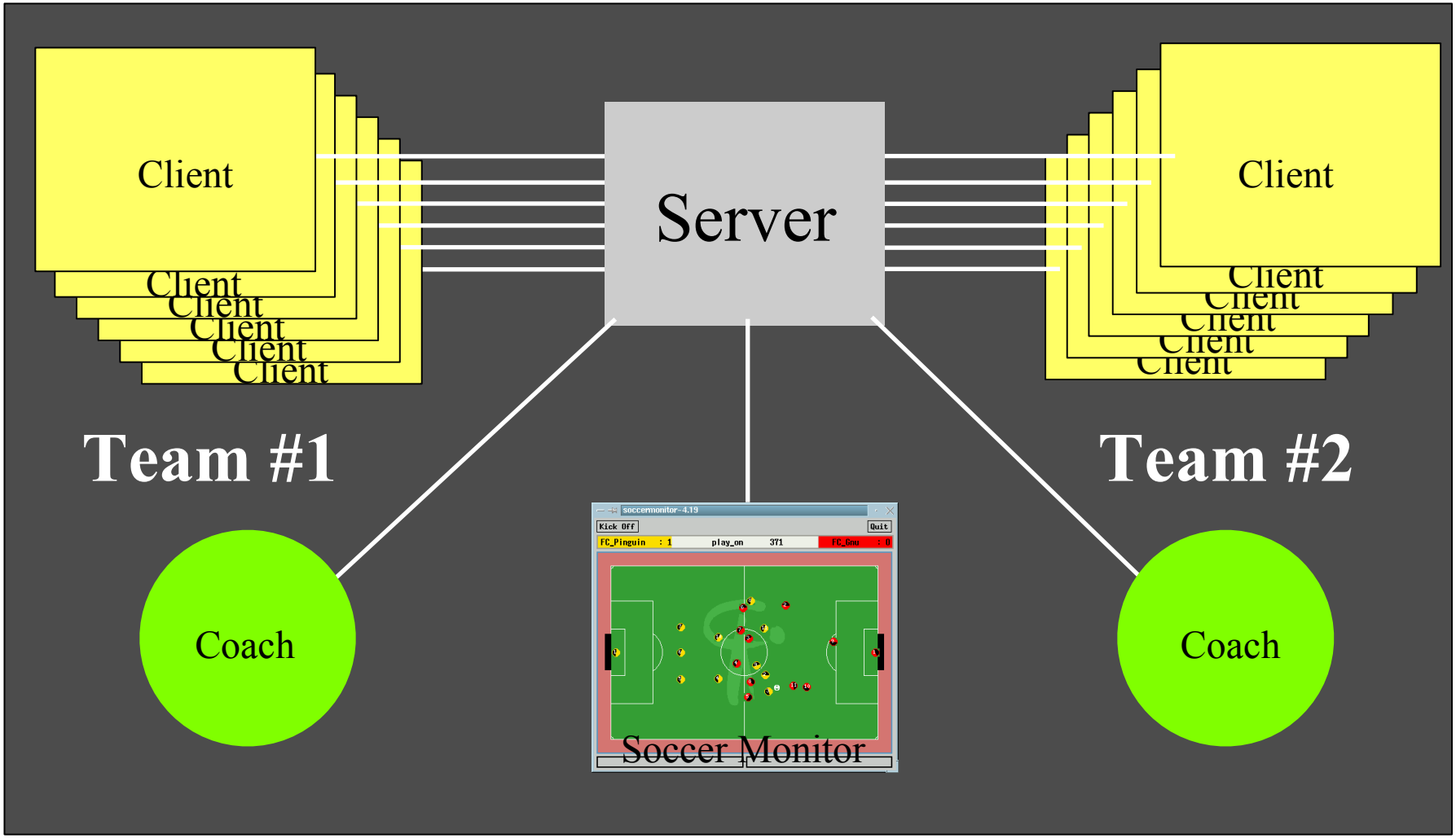
RoboCup Rescue:



RoboCup@Home:


Soccermonitor
0
Krislet
kick_off_l 29
Other
0
Bereit
NUM

Client-Server



Client

- Autonomous agents
- One agent represents one player
- Can be written in any language (C++, Java, Smalltalk, ...)
- Can be run on same machine or a network
- Clients may talk only to the server... not to each other!

Server

- “Referee” of the game and keeps time
- Maintains world model
- Tells agents what they can sense and handles agent actions

Starting a Game

- Download the software (http://www.nmai.ca)

1) Start the server

2) Start the monitor and connect it to the server

3) Start the clients and connect them to the server

4) Use the monitor to begin the game

Communication

decision-making
strategy

Sensory Input

Outputs

Connecting/Disconnecting

- From client to server
  - Connect
  - Reconnect
  - Quit
- From server to client
  - Confirms connection
  - Provides uniform number, side of field, state of game

Connection Example

- Client
  - sends connection message to server and asks to join myTeam
  - *init MyTeam*
- Server
  - Tells the player they are connected, have uniform #1, are on the right side of the field, and the game is pre-kickoff
  - *init r 1 before_kick_off*

Sensory Information

- Three main message types:
  - Hear – communication from other players
  - See – what is in their field of vision
  - Sense_Body – information about themselves
- Noise models for each

Hear Message

- Can hear one message per team per cycle
- Format: *hear Time [Direction] Sender "Message"*
  - Sender = online_coach_left/right, referee, self, or player
  - Direction (-180 – 180 degrees): where the sound came from
- Example:
  - *hear 408 -31 our 2 "Hello"*
  - At time 408, player 2 on our team said "Hello". The player was approximately -31 degrees from me.

See Message

- Format: *see Time ObjInfo*
- ObjInfo:
  - Type of object: ball, goal, line, flag, player
  - Parameters: distance, direction
    - Movable objects: change in distance/direction,
    - Players: body/head facing direction, team, uniform number,
    - Flags/Lines: location identifiers
    - Goal: side of field
- Each message can contain multiple ObjInfo
- Only distance/direction guaranteed, everything else just a bonus.

Example See Message

```
(see 18 ((f r t) 44.7 -22) ((f g r b) 47.9 30) ((f
  g r t) 42.5 13) ((f p r c) 30.3 34 -0 0) ((f p r
  t) 25.3 -7 0 0) ((f t r 40) 36.2 -37) ((f t r
  50) 44.7 -29) ((f r 0) 49.4 20) ((f r t 10) 47
  8) ((f r t 20) 46.5 -3) ((f r t 30) 48.4 -15)
  ((f r b 10) 53.5 30) ((f r b 20) 59.1 38) ((f r
  t) 44.7 -22) ((f g r b) 47.9 30) ((g r) 44.7 22)
  ((f g r t) 42.5 13) ((f p r c) 30.3 34) ((f p r
  t) 25.3 -7 0 0) ((f t r 40) 36.2 -37) ((f t r
  50) 44.7 -29) ((f r 0) 49.4 20) ((f r t 10) 47
  8) ((f r t 20) 46.5 -3) ((f r t 30) 48.4 -15)
  ((f r b 10) 53.5 30) ((f r b 20) 59.1 38) ((p
  "ExampleTeam") 36.6 28) ((l r) 41.7 -89))
```

- The right goal is at distance 44.7 and angle 22
- A player from ExampleTeam is distance 36.6 and angle 28

Sense Body Message

- (sense_body Time
  - (view_mode {high | low} {narrow | normal | wide})
  - (stamina StaminaEffort)
  - (speed AmountOfSpeed DirectionOfSpeed)
  - (head_angle HeadAngle)
  - (kick KickCount)
  - (dash DashCount)
  - (turn TurnCount)
  - (say SayCount)
  - (turn_neck TurnNeckCount)
  - (catch CatchCount)
  - (move MoveCount)
  - (change_view ChangeViewCount))

Sense Body Example

- *(sense_body 19 (view_mode high normal) (stamina 4000 1) (speed 0 0) (head_angle 0) (kick 0) (dash 0) (turn 0) (say 98) (turn_neck 0))*
- At time 19:
  - the player is using view mode high quality/normal width
  - has 4000 stamina left (and is exerting themselves at an effort of 1)
  - has no speed and is not moving in any direction
  - has their head facing straight
  - has performed no kicks, dashes, turns or turn_necks
  - is quite talkative and has said 98 things

Client Commands

Client Command	Once per Cycle
(catch Direction)	Yes
(change_view Width Quality)	No
(dash Power)	Yes
(kick Power Direction)	Yes
(move X Y)	Yes
(say Message)	No
(sense_body)	No
(score)	No
(turn Moment)	Yes
(turn_neck Angle)	Yes *

* can be used in the same cycle as catch, dash, turn, kick or move

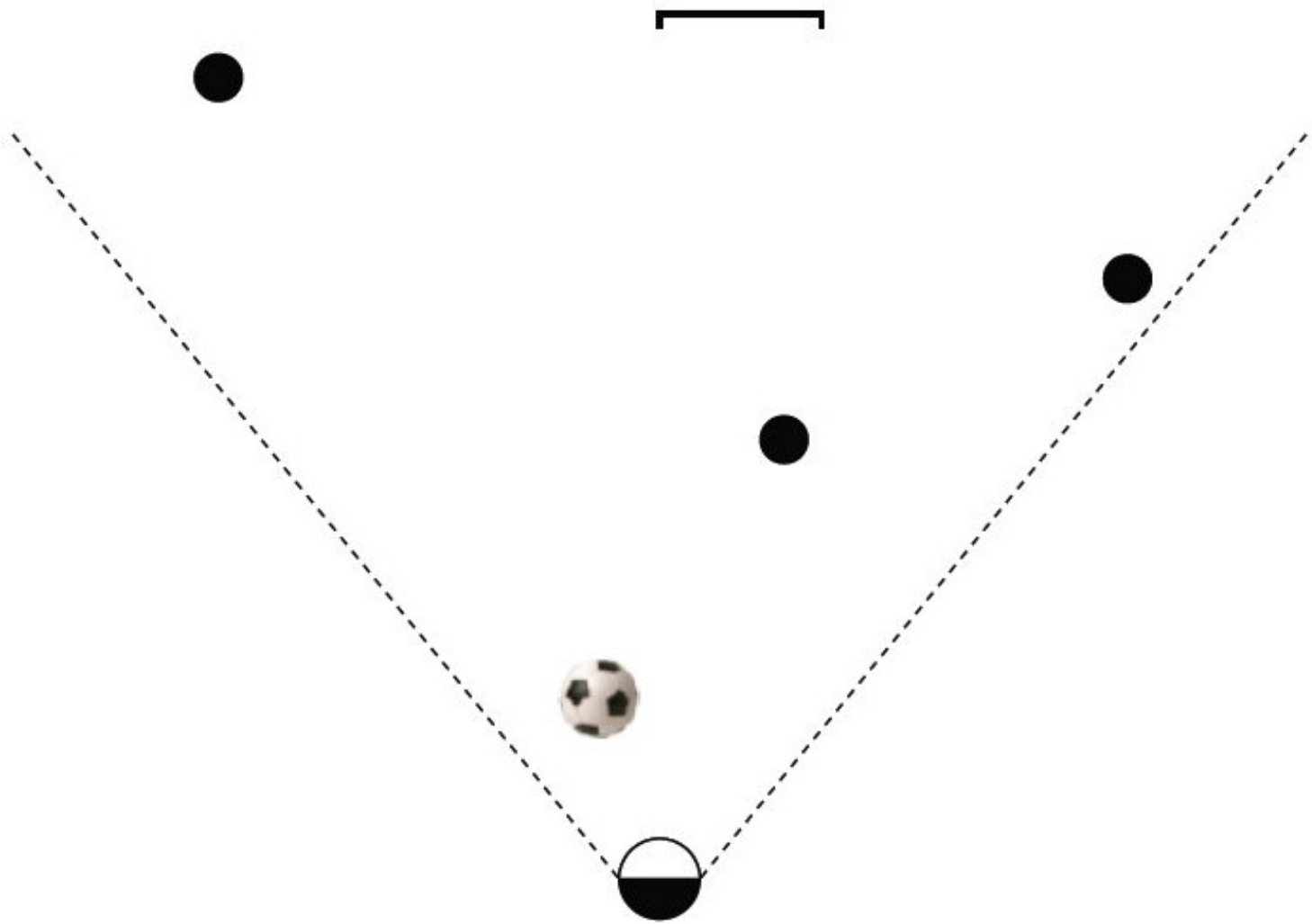
Command Examples

- say “message”
  - *say "Hello"*
- turn_neck angle
  - *turn_neck -5.97019*
- kick power direction
  - *Kick 100.0 41.0*
- dash power
  - *dash 82.0*
- turn direction
  - *turn 40.0*

The Environment

- accessible vs **inaccessible**: Only sees what is in front of it (with noise)
- deterministic vs **non-deterministic**: Just because agent wants to kick ball doesn't mean it will happen
- static vs **dynamic** : The players and ball will constantly be moving
- discrete vs **continuous**: Player can take any position on the field

Field of Vision



Krislet

- Modify the default Krislet behaviour by changing the **Brain.java** code.

- More specifically, modify the **run()** method

- You likely won't need to change much else

Brain run() method

```
public void run()
    {
     ...
     while( !m_timeOver ){
          object = m_memory.getObject("ball");
          if( object == null ){
               // If you don't know where is ball then find it
               m_krislet.turn(40);
          } else if( object.m_distance > 1.0 ){
               // turn to ball or if we have correct direction then go to ball
               if( object.m_direction != 0 )
                   m_krislet.turn(object.m_direction);
               else
                   m_krislet.dash(10*object.m_distance);
          }
          else {
             ... kick ball to goal ...
          }
     }
}
```

Brain run() method

```
public void run()
    {
     ...
     while( !m_timeOver ){
          object = m_memory.getObject("ball");
          if( object == null ){
               // If you don't know where is ball then find it
               m_krislet.turn(40);
          } else if( object.m_distance > 1.0 ){
               // turn to ball or if we have correct direction then go to ball
               if( object.m_direction != 0 )
                   m_krislet.turn(object.m_direction);
               else
                   m_krislet.dash(10*object.m_distance);
          }
          else {
             ... kick ball to goal ...
          }
     }
}
```

Brain run() method

```
public void run()
    {
     ...
     while( !m_timeOver ){
          object = m_memory.getObject("ball");
          if( object == null ){
               // If you don't know where is ball then find it
               m_krislet.turn(40);
          } else if( object.m_distance > 1.0 ){
               // turn to ball or if we have correct direction then go to ball
               if( object.m_direction != 0 )
                   m_krislet.turn(object.m_direction);
               else
                   m_krislet.dash(10*object.m_distance);
          }
          else {
             ... kick ball to goal ...
          }
     }
}
```

Brain run() method

```
public void run()
    {
     ...
     while( !m_timeOver ){
          object = m_memory.getObject("ball");
          if( object == null ){
                // If you don't know where is ball then find it
                m_krislet.turn(40);
            } else if( object.m_distance > 1.0 ){
               // turn to ball or if we have correct direction then go to ball
               if( object.m_direction != 0 )
                   m_krislet.turn(object.m_direction);
               else
                   m_krislet.dash(10*object.m_distance);
          }
          else {
             ... kick ball to goal ...
          }
     }
}
```

Brain run() method

```
public void run()
    {
     ...
     while( !m_timeOver ){
          object = m_memory.getObject("ball");
          if( object == null ){
               // If you don't know where is ball then find it
               m_krislet.turn(40);
          } else if( object.m_distance > 1.0 ){
               // turn to ball or if we have correct direction then go to ball
               if( object.m_direction != 0 )
                   m_krislet.turn(object.m_direction);
               else
                   m_krislet.dash(10*object.m_distance);
          }
         else {
            ... kick ball to goal ...
         }
     }
}
```

Brain run() method

```
public void run()
    {
     ...
     while( !m_timeOver ){
          object = m_memory.getObject("ball");
          if( object == null ){
               // If you don't know where is ball then find it
               m_krislet.turn(40);
          } else if( object.m_distance > 1.0 ){
               // turn to ball or if we have correct direction then go to ball
               if( object.m_direction != 0 )
                   m_krislet.turn(object.m_direction);
               else
                   m_krislet.dash(10*object.m_distance);
          }
          else {
              ... kick ball to goal ...
           }
     }
}
```

Resources

Software:

- http://www.nmai.ca - under Research Projects → Software Agent Imitation → Downloads
- **RoboCup Soccer Simulation Server and Monitor –** follow the link and download the recommended versions
- Also, the Krislet agent is a good place to start

- Server/Monitor/Krislet (versions used in demo) are available from course website.

Questions

Michael Floyd:

mfloyd@sce.carleton.ca