Real Time Autonomous System

PROJECT

SHOBHIT SHARMA ROLL NUMBER-MT19AI010

SIDDHARTH YADAV ROLL NUMBER-MT19AI011

Detailed Problem Specification:

We implement a simple A.I to play a 2D simulation soccer game. The simulator and the server are taken from The RoboCup Soccer Simulator. The server client we use is from https://github.com/jasontbradshaw/soccerpy.
Soccerpy handles the client-server communication and most of the backend details. This allows us to focus on writing the A.I. agent to play the game.

A team can have up to 12 clients, i.e. 11 players (10 _elders + 1 goalie) and a coach. The players send requests to
the server regarding the actions they want to perform (e.g. kick the ball, turn, run, etc.).
The server receives those messages, handles the requests, and updates the environment accordingly. In addition, the server provides all players with sensory information (e.g. visual data regarding the position of objects on the field, or data about the player's resources like stamina or speed).

Detailed design of the agents, collaboration and behavioural models :

The agents are:

1.reflex based.
2.utility based.
3.goal-based agents.

These agents aka. players (24) of the soccer game,plan and search for the optimal moves.

These optimal moves are of two types of behaviours:

1. locally:

 1.first…dribble

 2. then.. move

3.then …overtake nearby opponents

2. globally:

    1.first….pass.
    2.then…shoot.
    3.then….move

For the environment variables, we consider for each player agent:

1. the distance and angle to the goal post.
2. the distance and angle to the ball
3. the surrounding: the distance and angle of the nearest teammate
4. the surrounding: the distance and angle of the nearest enemy
5. the surrounding: is the path clear for ball, i.e. free of enemy interception
6. is the ball currently owned by our team
7. is the ball currently owned by the enemy team
8. is the ball kickable

The information will allow the use of our heuristics below. Then, we list the basic actions available to a player; these will be extended based on the role of the agent.

1. shoot (to the goal)
2. pass (to teammates)
3. move (run to strategic position, include dribbling)

Then, for the agent roles we striker, defender, and goalie. They implement variations of the actions above based on reflex (e.g. if a ball is shootable), utility (e.g. it is better to pass than shoot), and goal-based (e.g. scoring matters the most in the game).

Design of Dynamic interaction:

We have implemented 3 types of agent which are described below. The heuristics are given from the highest order of importance.

Striker (agent 1):

1. shoot at the goal post: if the ball is kickable, and agent is close to the goal post, and the path is clear to shoot. The agent finds the ball and the enemy goal post, and scan globally the shooting path is clear of enemies. It then aims at the enemy goal post, and calculate the optimal kicking power, and shoot the ball. This is the most important action as it scores the goal.

2. pass the ball: if this is the next-best course of action. The agent does so if it is blocked or approached by enemies. To pass, it scans for nearby ally with a clear shooting path, then calculate the optimal kicking power based on distance, and pass the ball. Passing is also more energy-efficient than dribbling and running.

3. dribble, if the above can’t be done, then the agent tries to dribble the ball closer to the goal post.

4. move to the ball, if enemy team currently owns the ball, and the agent is close enough to it.

5. move to defend, the striker moves back a little to get the ball back, if the enemy has the ball, and is close to our goal post.

6. move to enemy goal post, if our team has the ball. This is to prepare for attack and to accept a ball-pass from another teammate.

Defender(agent 2)

1. stay back, always try to remain on our side of the field to prepare to defend.

2. pass the ball, the same as from Striker agent.

3. dribble, the same as from Striker agent.

4. move to ball, the same as from Striker agent.

5. move to defend, the same as from Striker agent

Goalie(agent 3):

1. stay back, always try to remain close of the goal post to prepare for enemy striker.

2. pass the ball, the same as from Striker agent.

3. move to ball, when enemy is carrying the ball close to our goal post.

4. move to defend, the same as from Striker agent.

Implementation :

Description of the Software architecture of the system :

1.Download Soccer Server for simulation .

As per given in the problem we install version 16 of rcssserver from below link:

https://github.com/rcsoccersim/rcssserver/releases/tag/

2.Download for Soccer Monitor for visualisation.
As per given in the problem we install version 16 of rcssmonitor from below link:

https://github.com/rcsoccersim/rcssmonitor/releases/tag/
rcssmonitor-16.0.0

3.As we use MacOs Catalina version so the procedure for installing the above :

Steps:

1. Open terminal and install boost library : ->
Sudo port install boost

This will install boost and all its dependencies.

2.Compile and install rcssserver:

Go to directory where rcssserver 16.0.0 is unzipped.

->cd ~/(location)

Now configure the server by below command. This will build the necessary binaries to get you up and running. rcssserver is the binary for the simulator server.
The simulator server manages the actual simulation and communicates with client programs that control the simulated robots

So below is the command:

-> ./configure

->make

->sudo make install
3.install other packages for rcssmonitor:
-> sudo port install pkgconfig
-> sudo port install qt4-mac

If q4 is not compatible with ur os then u can opt for below command:

->brew install qt5

If u can’t access qmake then rectify this by running below linked command:

->brew link qt5 —force

4.Compile and install rcssmonitor:

Go to directory where rcssmonitor 16.0.0 is unzipped.

->cd ~/(location)
->*./configure*
->make
->sudo make install

5.

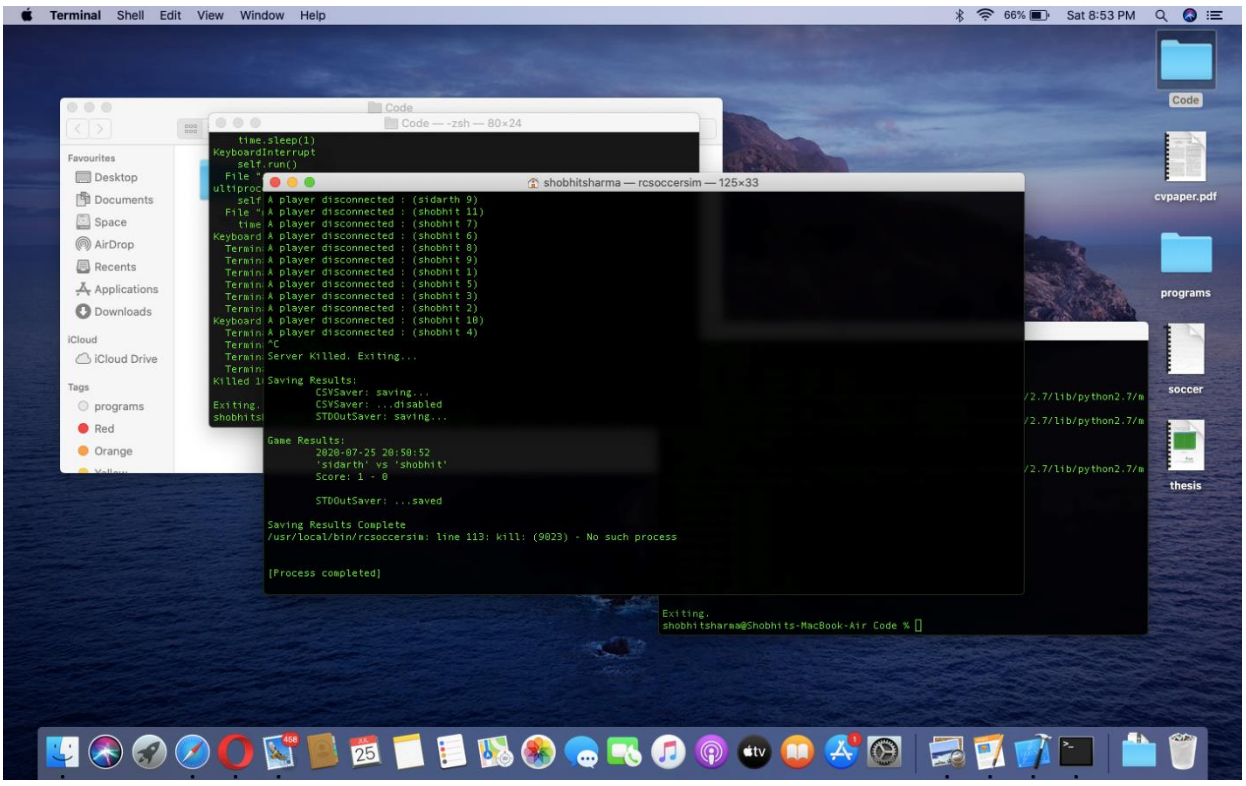
The default configuration will set up to install the simulator components in the following locations:

- /usr/local/bin
  for the executables

- /usr/local/include
  for the headers
- /usr/local/lib
  for the libraries.

6.rcsoccerism file covers both monitor and server
Run it from /usr/local/bin
->rcsoccerism

Sample output:
We ran the code for 9023 play ons the score board till then was siddharth 1-shobhit 0.Below is the screenshot of the terminal.



```
A player disconnected : (sidarth 9)
A player disconnected : (shobhit 11)
A player disconnected : (shobhit 7)
A player disconnected : (shobhit 6)
A player disconnected : (shobhit 8)
A player disconnected : (shobhit 9)
A player disconnected : (shobhit 1)
A player disconnected : (shobhit 5)
A player disconnected : (shobhit 3)
A player disconnected : (shobhit 2)
A player disconnected : (shobhit 10)
A player disconnected : (shobhit 4)
^C
Server Killed. Exiting...

Saving Results:
        CSVSaver: saving...
        CSVSaver: ...disabled
        STDOutSaver: saving...

Game Results:
        2020-07-25 20:50:52
        'sidarth' vs 'shobhit'
        Score: 1 - 0

        STDOutSaver: ...saved

Saving Results Complete
/usr/local/bin/rcsoccersim: line 113: kill: (9023) - No such process


[Process completed]
```

```
    time.sleep(1)
KeyboardInterrupt
    self.run()
```

```
/2.7/lib/python2.7/m
/2.7/lib/python2.7/m
/2.7/lib/python2.7/m

Exiting.
shobhitsharma@Shobhits-MacBook-Air Code %
```

Code with documentation for execution:

First, we enclose the soccerpy client code base into a module for exporting.

We make soccerpy/agent.py the basic agent class.\ Then we extend this base agent class for the 3 agents described above.

We modify in soccerpy the modules agent, world model to allow for the use of our heuristics.

For example, we added the methods to get teammate, enemy, scan the agent's surround- ing for enemy v.s. teammate, determine if the target path is clear from enemy interception, who owns the ball, distances among objects. The main.py imports these 3 agents, initialize 5 strikers, 5 defenders, and a goalie, then set up the positioning.

For execution follow steps:

```
->/usr/local/bin/rcsoccerism

->cd (location of code)

->python main.py
```

All players are set in the field with the TeamName given in main.py file

Now change the TeamName with the opponent team and Open again terminal window :

```
->cd (location of code)

->python main.py
```

All players are set in the field with the opponent TeamName given in the main.py file

All are set and game is running between two teams