Playing with *Agent_SimpleSoccer*

1. Start the server
2. Start the agent(s) in NetBeans
3. Start the game: press “k” and “b” in the monitor window

(see below for details)

1. *Start SimSpark Soccer Server (the simulation of the environment):*

   goto

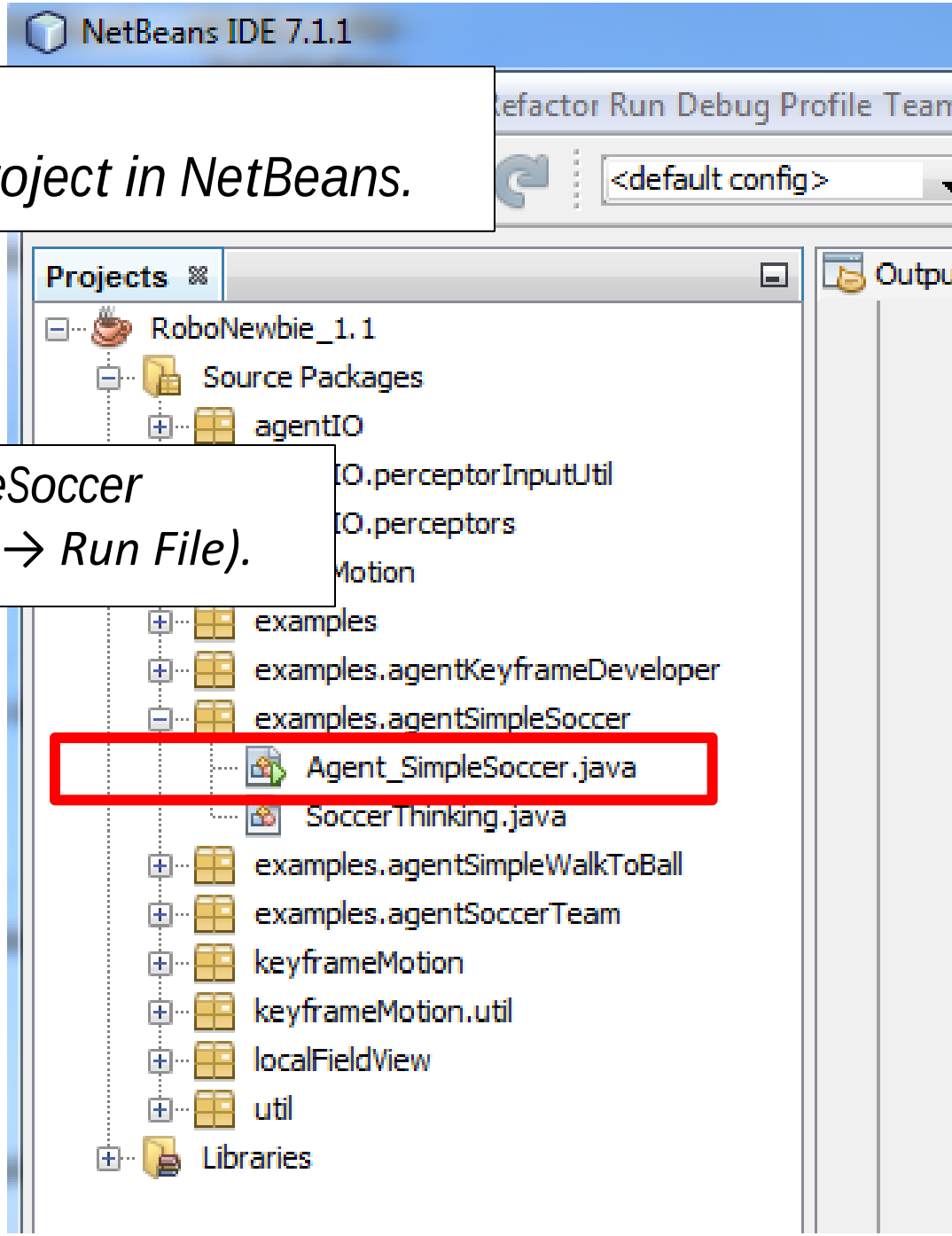
   start the Server

The monitor window with the Soccer field appears.

You can navigate at the monitor window with
left mouse button, arrow keys, keys a,s,d,w, page up, page down.

2. *Start a Player:*
   Open the RoboNewbie-Project in NetBeans.

Run the class Agent_SimpleSoccer
(right click on the filename → Run File).



A player appears on the field. Its team name is seen at the top. It starts moving. But it is not allowed to cross the middle line by the soccer rules before kick off (it leads to a crash of the server).

Hence, you (as the human referee) have to start the game:

***Press "k"** and **"b"** in the monitor window*

Then the game state at the top of the monitor changes to "Play On" and time starts running.

The identity of a player is defined by player number and team name.

```
70        the robots on the field have either rec
71        robot has grey parts. */
72      final String id = "1";
73      final String team = "simpleSoccer";
```

Its initital pose is defined by the beam coordinates.

Position: (beamX, beamY)
Orientation: beamRot

```
78        half. The robot can be placed
79        variable beamRot, in degrees,
80      final double beamX =    -0.5;
81      final double beamY =     0.4;
82      final double beamRot =   -40;
83
```

beamX <0 is required (since players start in their own half)

You can change all these values to make further programs for further players and teams.

Whenever you start the server, the first started team becomes the left team with color blue. The second team must have a different name. It becomes the right team and gets color red.

The implementation of *Agent_SimpleSoccer*

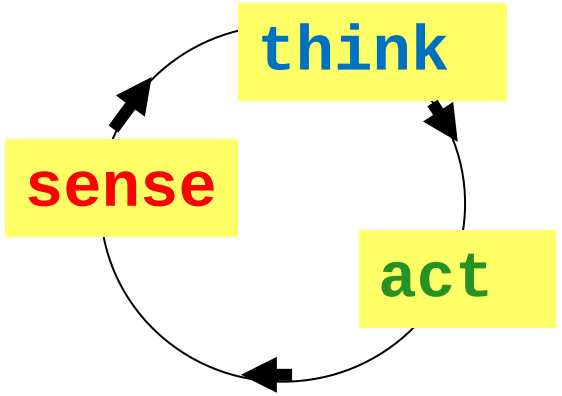
- Player identity (as described above)
- Method init: Initialization: Connect to the server
- Method run: Sense-think-act loop
- Method sense: Get perceptor data via IO-packages
  Synchronize with the 20msec server cycle
- Method think: Decide for next motions
  Implemented in class SoccerThinking
- Method act: Continue with active motion
  Send effector message via IO-packages

```
public void run(){   ….
   for (int i = 0; i < totalServerCycles;  i++)
   {   sense();  think();   act();
   }
}

 private void sense() {
    percIn.update(); localView.update();
 }

private void think(){
    soccerThinking.decide();
 }

 private void act(){
   kfMotion.executeKeyframeSequence(); lookAround.look(); effOut.sendAgentMessage();
 }
```



The implementation of *SoccerThinking*

Idea of the program:

Repeat (whenever a motion is complete):
- If robot has fallen down: Stand up
- If position of ball is not known:
  - Search for ball by turning head (and body)
- else if if ball is far away: turn to ball, walk to ball
- else if ball not between player and goal: turn around ball
- else walk forward („dribbling“)

The implementation is very simple – what happens?
What could be improved?

```
public void decide() {
if (motion.ready())                          // decide for new motion only if last motion completed
if (percIn.getAcc().getZ() < 2) {    … stand up …    }              // robot has fallen down
else if ((serverTime - ball.getTimeStamp()) < lookTime)          //  robot has ball coordinates
    if (Math.abs(ballCoords.getAlpha()) > TOLLERATED_DEVIATION)
          { ….  turn towards the ball ….}                                  // ball not in front of robot
   else if (ballCoords.getNorm() > TOLLERATED_DISTANCE)
          { …   walk towards the ball …  }                                // robot far away from ball
   else if ((serverTime - oppGoalLPost.getTimeStamp() < lookTime)  && ….)
                                                                          // robot has goal coordinates
            if ((oppGoalLPost.getCoords().getAlpha() <= ballCoords.getAlpha()) ||  ….)
            {   … sidestep for better position ….   }                  // ball not  between robot and goal
           else   {  …  walk forward …   }                             // robot in good dribbling position

        else  { … turn to see goal …}                                 // robot does not have goal coordinates
else   { … turn to see ball …}                                        // robot does not have ball coordinates
 }
```

Playing with *Agent_SoccerTeam*

Agent_SoccerTeam allows you to have only one program for players with different roles and different inititial positions.

If the player id is set to "0“ in Agent_SoccerTeam, each call of the program instantiates a new player. The player numbers are then assigned by the server in the order of starting the programs.

```
80          identity.
81          */
82         final String id = "0";
83         final String team = "SoccerTeam";
84
```

For a second team you can use the same code if you specify another *final String team = "<name of second team>"*;

The different inititial poses for players with player id = "0" are defined in the class util.Beamposes.

```
public class BeamPoses {

  private static final double [][] Poses =
 {
        {-5, 0.0, 0.0}, //player 1, take (-5.0, 0, 0) for goalkeper
        {-0.5, 0.0, 0.0}, //player 2
        {-1.0, 2.0, 0.0}, //player 3
        {-1.0, -2.0, 0.0}, //player 4
        {-3.0, 0.0, 0.0}, //player 5
        {-3.0, -1.5, 0.0}, //player 6
```

-4.5 might be better

Depending on their numbers, the players can have different roles.

The different roles are specified in separate classes.

They are assigned to players according to the player id

(see class *SoccerTeamThinking*):

```
116             */
117             switch (playerNumber) {
118                 case "1":
119                     role1 = new SimpleGoalie(motion, percIn, log);
120                     break;
121                 case "2":
122                     role2 = new SimpleSoccer_Plovdiv2014(motion, percIn, log);
123                     break;
124                 case "3":
125                     role3 = new SimpleSoccer(motion, percIn, log);
126                     break;
127                 case "4":
128                     role4 = new SimpleSoccer_withKick(motion, percIn, log);
129                     break;
```

Implementation of a player with a specific role is done in 3 steps:

1. Implement a class for the intended role
   with a related *decide* method.
   Examples are found in the package *examples.agentSoccerTeam*

2. Integrate the role in class *SoccerTeamThinking*
(see next slide)

3. Define the start position in class util.*BeamPoses*

2. Integrate the role in class *SoccerTeamThinking*

at three places (using the intended player number):

- Declaration

```
SimpleGoalie role1;
...
```

- Instantiation

```
switch (playerNumber) {
        case "1":
            role1 = new SimpleGoalie(motion, percIn, log);
            break;
       ...
```

- Call of decide method

```
public void decide() {
      switch (playerNumber) {
          case "1":
              role1.decide();
              break;
        ...
```

Game states (play modes):

The actual state of a game is shown in the monitor window.

Possible states are listet in util.GameStateConsts:

```
public static enum PlayMode {
   BeforeKickOff,
   KickOff_Left,
   KickOff_Right,
   PlayOn,
   KickIn_Left,
   KickIn_Right,
   CORNER_KICK_LEFT,
   CORNER_KICK_RIGHT,
   GOAL_KICK_LEFT,
   GOAL_KICK_RIGHT,
   OFFSIDE_LEFT,
   OFFSIDE_RIGHT,
   GameOver,
   Goal_Left,
   Goal_Right,
   FREE_KICK_LEFT,
   FREE_KICK_RIGHT,
   NONE }
```

- Game states are set by the built-in referee according to the situation (e.g. corner kick). Then players can act according to soccer rules (as far as related behavior is implemented).

- Human referee can set some game states by klicking in the monitor window:
  - k: kick-off
  - b: drop-ball (for play-on in our distribution)
  - l/r: free-kick left/right

  For more information see: SimSpark-long.pdf .

- You can implement related behaviors for different situations (e.g. for corner-kick).
- They must be called in Agent_SoccerTeam

  (cf. method think and method act) .