```
* Team informations:
********************

Team Name     : Cow_n_Action
Country       : Belgium
Instituton    : ULB
Team's leader : Nicolas Glady
email         : nglady@ulb.ac.be

* General overview:
*******************

- based on the CMUnited98 code for low-level functions
- project made for a thesis and should be overtaken and improved year by year
- use of a simplified production rule system for clarity:
  - preconditions are checked in order to apply a rule
  - macroactions which contain the code to be executed if its preconditions are matched

  - costs to retain the best action if the preconditions of more than one action are matched

- neural networks used for complex and non-linear functions/behaviour (not yet finished)

- raytracing-like method used to determine the best angle to shoot/pass

* Production rule system:
*************************

As this work should be overtaken by students of our university year by year, we thought it

should be easier for our successors to cut the code into small functions which are more

understandable than large and complex routines. Each of those small functions (called

macroactions) possess:
- preconditions (a list of flags describing the game's state) in order to run its corresponding

  action
- a list of flags to specify in which game's state the action should be interrupted

- a list of modified flags by the action (as the game's state and the flags describing it

  change through the actions we made) which are used for planification of further actions to

  make in the future
- as more than one action may be taken at the same time, a cost function is needed to

  determine the fitness of those actions and keep the best one
- the code of the choosen action

syntax:

MACRO_ACTION(<action_name>, <on_screen_action_name_for_debugging>)
PRE_CONDITION(<flag_name>, <flag_value>)
STOP_CONDITION(<flag_name>, <flag_value>)
MODIFY_FLAG(<flag_name>, <flag_new_value>)
COST
{
  // calculate the cost of the action here
}
MACRO_BODY_START
{
  // execute the code of the action here
}
MACRO_BODY_END

simple example:

MACRO_ACTION(run_for_ball, "RUN FOR BALL")
PRE_CONDITION(FASTEST_ON_BALL, true)
PRE_CONDITION(PLAY_MODE_ON, true)
STOP_CONDITON(BALL_KICKABLE, true)
COST
{
  return PredictedCyclesToPoint(InterceptBallPosition(SP_max_power), SP_max_power);
```

```
}
MACRO_BODY_START
{
  RunToPoint(InterceptBallPosition(SP_max_power), SP_max_power);
}
```

As the actions differ from players to players (for example, strikers won't have the same behaviour as defenders), each type of player (producer) has a list of macroactions he can perform. Orders sent by that type of player can also be defined here.

syntax:

```
PRODUCER(<producer_name>)
RULE(<action_name>)
ORDERS
{
  // orders to be sent
}
END_OF_PRODUCER
```

simple example:

```
PRODUCER(Goalie)
RULE(scan_field)
RULE(go_catch_ball)
RULE(replace)
RULE(shoot_ball_away)
RULE(catch_ball)
ORDERS
{
  // if we are in a defense phasis, ask a teammate to hold closest opponent
  Unum opponent = ClosestOpponentToPoint(MyPos());
  Unum teammate = ClosestTeammateToPoint(OpponentAbsolutePosition(opponent));
  AskTeammate(teammate, opponent, HOLD);
}
END_OF_PRODUCER
```

This is clearly the crucial point of our team even if it doesn't really have to do with AI but this was needed for transmitting a clear code to our successors year by year... We hope they should be able to focus over other aspects of the simulation during the next years.

* Raytracing:
****

This method is used to see which angle is the safest for shooting, passing or which one is the most unsafe for positionning the goalie where opponents have the biggest angle to score. In facts, we look for the largest beam of good angles and take the middle angle of this beam.

algorithm overview for shooting in opponent goal:

```
AngleDeg ray_angles[RAY_DISCRETISATION];
bool     ray_bool[RAY_DISCRETISATION];

// init the ray_angles to range from angle(MyPos2LeftGoalMarker) to angle(MyPos2RightGoalMarker)


for (int i = 0; i < RAY_DISCRETISATION; i++) {

  Ray ray(MyPos(), angles[i]);
  ray_bool[i] = true;

  for (int j = 1; j <= SP_team_size && ray_bool[i]; j++) {
    if (OpponentPositionValid(j)) {
      ray_bool[i] = !ray.IsInRightDir(OpponentAbsolutePosition(j));
    }
  }
}
```

// now, look for the largest beam of true values in ray_bool and take the middle angle matching

* Neural Network:
****

As some functions are complex to evaluate in the simulation, we have thought of using neural networks like perceptron for approximation. But as we write this doc, some convergence's problems still remain =(We hope to be ready for the tournament in June (if qualified of course) and have planned to use genetic algorithm to find closest optimal solution. The structure we are currently using is a 3 layers perceptron (1 input layer - 1 hidden layer - 1 output layer) with backpropagation for computing new weights of connections but this may change if we find a better structure (multi-hidden layers). Those networks will be used for interceptions, kicks, dribbles...

* Conclusion:
****

The techniques described here are very classical methods used by other teams. As we do have other obligations in life, we couldn't work full-time on this project. Yet, we are not very numerous to work on the Robocup as our University (well, most of our professors) is very unlikely willing to employ full-time researchers to work on it. Maybe, we're going to establish contacts with other Universities willing to join a multi-universities virtual soccer team... Without a critical mass of 4-5 persons working on it, we won't be very competitive in the next years.

Some points are still to be improved:

- new attacks should be made (for now, only simple attack phases have been made like quick counter-strike) but defense is rather good
- neural networks should be finished to improve precision of kicks, interceptions... we are currently using analytical functions until convergence's problem are fixed.