**TERM PAPER**

**ARTIFICIAL INTELLIGENCE**

**Implementing RoboCup Soccer Defense Strategy**

**Version 2.0**

**Date- 16th November,2014**

**Outline of the Problem , Introduction to the Strategy and Justification**

Implementing defense strategies means that we would be dealing with understanding of not having the ball with us and also devising a strategy that would stop the opposite teams from scoring goals. We need to design the strategy in such a way that we are only concerned with not letting the opposition score but also in such a way that the defense does not hold up the play.

Modern Football has evolved and normally a hybrid of man-to-man marking and zone marking is used in defense. The extent of hybridization depends upon us. The Football field is an unknown environment and is in fact not deterministic . In this paper we will discuss.

In our project we will be using a form of zonal defense, defensive players mainly move in relation to each other, whereas a man-to-man defense mainly moves in relation to opposing players.

So to sum it up, our aim will be not only to save goals but also to move the ball forward and not holding up the play for too long. It is to note that we are only implementing defense but will surely like to avoid holding the play.

Decision making is highly required to take and make precise decisions. For eg the decisions can be when to force clear a ball, when to smart clear (try and find a player while clearing), when to dash for the ball and when to put your body in the way etc.

We also would require our agents to learn and adapt on their own. Methods of reinforced learning can be applied here. However this paper will only contain basic strategies and the codes we have implemented so far.

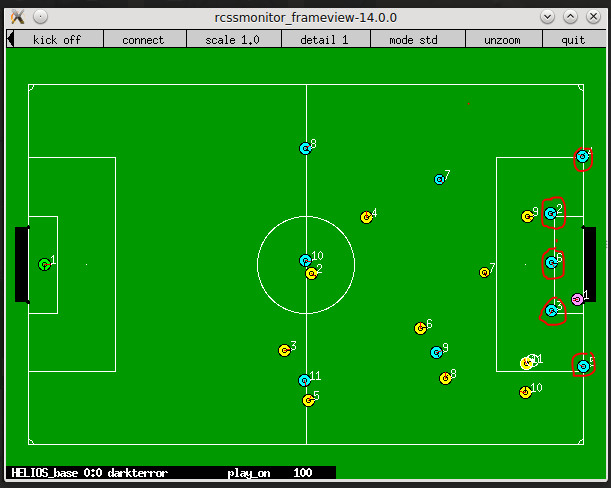
In the coming sections we will cover the philosophy behind our Strategy.

**Approach- Methodology and Algorithms**

Motivation for Zonal defence :-Zonal marking can be applied through the game. The basic philosophy is assigning each player with a designated area to protect. However there is always a weakness when it comes to playing on the perimeters of adjacent zones. This may cause confusion and players may leave large spaces of field open.

Approach to our Strategy basically started with identifying the entry points of attack .

We then tried positioning the players through a static call to a function called defencemove() which placed the players at the entry points.



However getting the positioning right is one part of defence, we also need to add functions that will chase the ball ,intercept and tackle accordingly. So a function called chaseball() with the following functionalities was added.

A soccer environment basically has four stages regarding the play.They are-

1. Opponent has the ball -This is the part where the opponent has the ball and our functions for defence will be called.
2. Common Ball- Since both us and the opponent have equal chance of getting the ball,attempt a tackle or do a ForceKick.
3. None have ball- set our Body and Neck towards the Ball.
4. We have Ball- We implemented basic move functions here so that our team doesn’t give up the ball when we get the ball close to our goal.

**Defence Algorithms**

**DefenceMove -**  In this strategy we try to bring the player at their own position by just statically assign them the zone. This strategy turns out to be static as in even if other player is available, the player we specified will only move to that place.Here we have defined the parameters according to the server pitch(ground) provided to us.

**Clear Ball**-

Attempt a tackle;

If(tackleSuccessful)

move with ball;

else

ForceKick the ball;

**Chase Ball**- Makes a player dash for a ball. So if the agent is not facing the ball, we first turn the agent towards the ball and then dash towards the ball and we also need to find the closest agent to the ball. In this code we used that concept of intercept table as we will find the distance with number of **cycle required by agent to receive to that ball** const int self\_min = wm.interceptTable()->selfReachCycle()

const int mate\_min = wm.interceptTable()->teammateReachCycle();

const int opp\_min = wm.interceptTable()->opponentReachCycle();

Now we will compare the parameter like :-

if ( ! wm.existKickableTeammate() && ( self\_min <= || ( self\_min <= mate\_min && self\_min < opp\_min + 3 ) ))

{

agent will intercept

return true

}

Then we will find parameters like thr\_distance, dash\_power to make the agent move to the ball

const Vector2D target\_point = Strategy::i().getPosition( wm.self().unum() );

const double dash\_power = Strategy::get\_normal\_dash\_power( wm ); //dash power of agent to move to that point

double dist\_thr = wm.ball().distFromSelf() \* 0.1;//distance of the ball from self

if ( dist\_thr < 1.0 ) dist\_thr = 1.0;

if(agent can’t reach the ball)

{

face the ball;

}

if(kickable opponent && agent dist < 15.0)

{

tackle ball;

}

else

{

face the ball;

}

**ClosestPlayerToZone**- We specify five zones we want to mark. A static point for each zone is taken as the reference point for calculating distance from players. The closest player from that reference point of the particular zone is assigned that zone. Note that this process is dynamic.

This ensures the best fall back for defence.

We declare globally five player numbers as rb,rcb,cb,lcb,lb which will be assigned zones.

if(OpponentJustGotBall)

{

rb=FindClosestPlayerToZone(z1);

rcb=FindClosestPlayerToZone(z2);

cb=FindClosestPlayerToZone(z3);

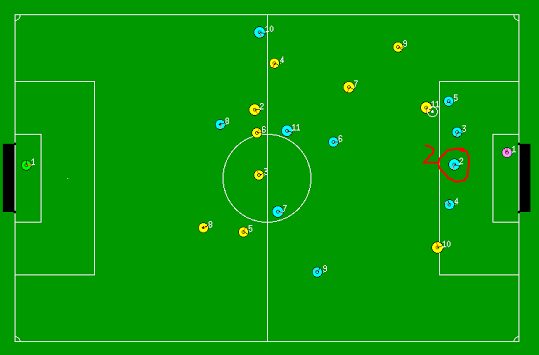
lcb=FindClosestPlayerToZone(z4);

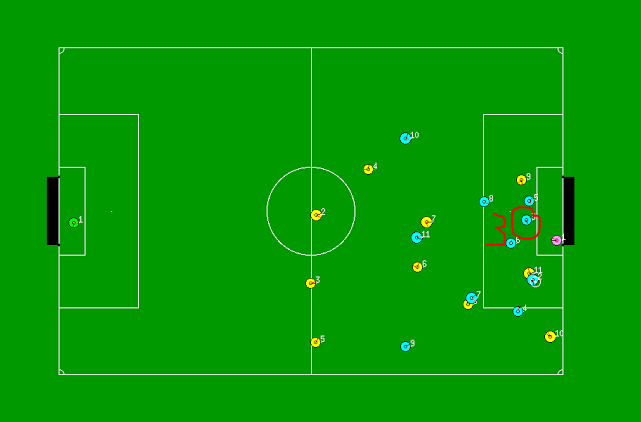
lb=FindClosestPlayerToZone(z5);

}

Move the player to with this number to the particular defending positions in their zones;

Note- The player numbers for each zone will be computed only when the opponent has the ball and only once.





**FindClosestPlayertoZone**- In this function we are just finding the closest player to zone and just giving the player number as the output. This function has five arguments agents,x-position,y-position,then two arguments are the boundary case for the zone. Basically we are defining the zone and breaking the defence are in 5 parts. In this function we will iterate through number 2 to 11.

for(int i=2; i<=11; i++){

if(agent->world().ourPlayer(i)!=NULL)

{

if(player(i) position is in between buffer1 and buffer2 and player distance from point(x,y) is less than leastdistance )

{

leastdistance=thisplayer(i).distancefrompoint;

}

count = i;

}

return count; this function will return the player number after iterating through all the player.

**Fall- Back** – Fall back is called when the opponent has the ball. Fall back is a combination of ClosestPlayerToZone and Chaseball. So basically we are coming back to our zones and trying tackles and intercepts on the way.

fallback()

{

closest player to zone move towards defined zones;

chase the ball;

}

**Future Scope**

As we know Artificial Intelligence is all about making the agents learn on their own and using techniques to do efficient Machine Learning.

Let us Consider our Dynamic implementation of Zone assignment. To avoid repeated iterations we can implement an array of size 10 for each zone. Each element is set to zero. The index of the array will represent the player number.

Example-

Let RightBack(rb) be an array.

Each time right back is assigned a player number rb(player-1)++;

WE can do this till halftime.(Learning Period)

After Half Time we can statically assign positions to specific players according to the index that has the highest value. This saves a lot of iteration cycles.