实验七

关于分工，敬成超，屈天奇，周诗奥都贡献了自己的力量。一开始，由周诗奥上网搜集资料。屈天奇，敬成超负责实现具体的要求细节。明确了修改方向，鉴于我们水平初级，便想在一些细节入手，让踢球更人性化，更有竞争力。敬成超负责整合意见，撰写报告。

于是，我们就开始实践了。

敬成超在basicplayer.cpp里增加了一些函数，让球员的动作更多样化，抉择更灵活。

以下是他的修改代码，及见解。

*// 朝较大的夹角射门*

根据与守门员的角度关系，选择一个适当的位置进行射门。

如果角度差异较大，选择向一个侧边射门，从而可能提高得分的概率。

提高了机器人射门时的智能化，能够更灵活地选择最佳的射门方向，尤其在面对强手时，能避免守门员拦截。

SoccerCommand BasicPlayer::maxAngShoot( VecPosition posAgent )

{

SoccerCommand soc;

VecPosition posGoalie;

AngDeg ang\_goalie, angup, angdown;

posGoalie = WM->getGlobalPosition(WM->getOppGoalieType());

ang\_goalie = (posGoalie - posAgent).getDirection();

angup = (VecPosition(52.5,6.0) - posAgent).getDirection();

angdown = (VecPosition(52.5,-6.0) - posAgent).getDirection();

if(fabs(angup - ang\_goalie) > fabs(angdown - ang\_goalie))

{

soc = kickTo(VecPosition(52.5, -6.5),2.7);

}

else

{

soc = kickTo(VecPosition(52.5, 6.5),2.7);

}

return soc;

}

*// 朝球门方向将球以最大力度踢球*

：

保证踢球的力度达到最大，使得球更难被守门员拦截，增强了直接得分的机会。

SoccerCommand BasicPlayer::maxAngShoot1( VecPosition posAgent )

{

SoccerCommand soc;

VecPosition posGoal(PITCH\_LENGTH/2.0,

(-1 + 2\*(WM->getCurrentCycle()%2)) \*

0.4 \* SS->getGoalWidth());

soc=kickTo( posGoal,SS->getBallSpeedMax());*// 朝球门方向将球以最大力度*

return soc;

}

*// 闪避带球*

处理带球时的闪避逻辑。

计算现在的运动方向，再根据敌方位置判断是否需要调整运动方向来躲避对手。

让机器人在带球时更具反应能力，能够在对抗中灵活应对，降低被对手拦截的风险

SoccerCommand BasicPlayer::dribble\_Dodge(VecPosition tar, VecPosition posAgent)

{

SoccerCommand soc;

AngDeg ang = ( tar - posAgent ).getDirection();

soc = dribble(ang, DRIBBLE\_FAST );

if( WM->isOpponentAtAngleEx( ang - 45, ang , 7 ) )

{

ang += 45;

soc = dribble( ang , DRIBBLE\_FAST );

}

else if( WM->isOpponentAtAngleEx( ang, ang + 45 , 7 ) )

{

ang -= 45;

soc = dribble( ang , DRIBBLE\_FAST );

}

return soc;

}

*// 计算角球应该踢向的点*

**4.** 计算角球应该踢向的目标位置。

通过当前球的位置，计算并返回最佳的踢向目标位置，以增加进攻时的灵活性。：

提供了明确的踢向点，有助于在比赛中进行更精准的策略执行，提升了团队配合的效率

VecPosition BasicPlayer::pointKickToC()

{

VecPosition posBall, posKickTo;

double x, y;

posBall = WM->getBallPos();

x = posBall.getX();

y = posBall.getY();

posKickTo.setX(- x / fabs(x) \* 5 + x );

posKickTo.setY(- y / fabs(y) \* 12 + y );

return posKickTo;

}

*// 计算界外球应该踢向的点*

通过当前球的位置，计算并返回非被防守位置的最佳界外球踢向目标。

确保机器人在界外球时能够有效地给队友创造机会，增强比赛中的策略灵活性。

VecPosition BasicPlayer::pointKickToK()

{

VecPosition posBall, posKickTo;

double x, y;

posBall = WM->getBallPos();

x = posBall.getX();

y = posBall.getY();

posKickTo.setX( x + 7 );

posKickTo.setY(- y / fabs(y) \* 12 + y );

return posKickTo;

}

这些修改和新增的功能整体上增强了机器人的实战能力，使得它们可以在动态环境下更好地做出决策。通过智能化的射门、强劲的踢球、灵活的闪避、清晰的传球点计算，机器人在比赛中的表现得以提升。这样的战术优化使得机器人在复杂场景下仍能做出有效的反应，并保持游戏的竞争性。而通过这些代码的方法，团队合作和策略执行也变得更加协调和有效。

周诗奥同学的代码，在WorldModel.cpp里做了修改，

void WorldModel::setSynchSeeMode()

{

bSynchSeeMode = true;

}

bool WorldModel::isSynchSeeMode()

{

return bSynchSeeMode;

}

第一段代码新增了setSynchSeeMode和isSynchSeeMode函数，用于设置和检查是否处于同步视觉模式。

bool WorldModel::ifCanPassBall(Robot robot)

{

*// ... 省略部分代码 ...*

*//搜索是否存在传球路径*

for(int i=1;i!=5;++i)

{

*// ... 省略部分代码 ...*

if(ifOpsCannotIntercept(preBall[countBall].getPosition()))

{

if(ballTime >= ownTime)

{

*//把截球相关信息存入结构体*

tempInter.pos = preBall[countBall].getPosition();

tempInter.robotNo = i;

tempInter.time = ballTime;

tempInter.ang = ang;

tempInter.value = 0;

pathVector.push\_back(tempInter);

DoLog(CT\_TEST, "执行了信息压入！");

}

else

{

continue;

}

}

else

{

continue;

}

}

*//判断是否存在传球路径*

if(pathVector.size() > 0)

{

return true;

}

else

{

return false;

}

}

ifCanPassBall函数的逻辑有所修改，特别是在判断是否可以传球时，考虑了对方是否无法拦截球的情况。

void WorldModel::DoorKeeper(void)

{

*// ... 省略部分代码 ...*

if(ifCanCatch())

{

cmd.CommandType = Com\_Catch;

ownRobots[0].setCommand(cmd);

}

else if(ownRobots[0].getPosition().getDistanceTo(ball.getPosition()) < CKICKABLEDIST)

{

if(ifCanPassBall(ownRobots[0]))

{

passBall(ownRobots[0], doorkeeperCmd);

ownRobots[0].setCommand(doorkeeperCmd);

DoLog(CT\_TEST, "踢球方向 %f, 踢球力度 %f", doorkeeperCmd.ComParam0, doorkeeperCmd.ComParam1);

}

else

{

kickOutOfArea(ownRobots[0],doorkeeperCmd);

ownRobots[0].setCommand(doorkeeperCmd);

}

}

else

{

*// ... 省略部分代码 ...*

}

}

第一段代码中，DoorKeeper函数的逻辑有所修改，特别是在守门员决策时，考虑了是否可以传球和是否可以将球踢出禁区的情况。